| CSCI-GA.3210-001<br>MATH-GA.2170-001 | **Introduction to Cryptography** | January 25, 2012 |
|---|---|---|

# Lecture 1

*Lecturer: Yevgeniy Dodis* *Spring 2012*

These notes define the problem of secure communication, identify the relevant agents, and offer some assumptions about those agents. An initial effort at a solution involves the agents sharing a secret, and produces a private-key method called the One Time Pad algorithm, which seems a good candidate for secure communication. To check if this is true, the notion of perfect security is then defined more precisely and OTP is proved to be secure. Although OTP is secure, it has several undesirable features, such as unbounded key lengths. (We also prove that no secret key cryptosystem is secure in the Shannon sense unless the key length is unbounded. This is the Shannon Theorem.) We attempt to circumvent the key length problem by changing the assumptions about the agents. Namely, we no longer assume a computationally unbounded adversary. This leads to a new class of "public key" encryption methods where the agents do not share a secret. The requirements of public key encryption motivate the use of One Way Functions and trapdoor functions, whose properties are described.

## 1 Description of Problem

Assume there are three agents, Bob, Alice, and Eve. Bob wants to send Alice a private letter that only Alice can read. Eve is an adversary who may intercept the letter, but reading it should not enable her to reconstruct Bob's message to Alice. This is the essence of the problem of secure communication. A more formal definition of "secure" will be given later.

To meet these requirements, Bob must somehow alter his message to Alice so it cannot be understood by anyone else. This alteration is called encryption. If $m$ is the message to be encrypted, (also known as the "plaintext" or the "cleartext") then $c$, the encrypted message, (also known as the "ciphertext") is produced by an encryption function $E(m, ???) \rightarrow c$, which Bob uses. Alice uses a symmetric decryption function $D(c, ???) \rightarrow m$.

The question marks in the function definitions stand for other pieces of information that may be provided to the encryption/decryption functions, depending on the specific method being used. Some possibilities are:

- $S_b$: A piece of secret information known to Bob but not to Eve.
- $S_a$: A piece of secret information known to Alice but not to Eve.
- $PK$: A piece of public information, available to everyone.
- $R$: A random factor, presumably drawn from some pseudo-random distribution. This variable is only applicable on Bob's end; Alice deterministically decrypts the ciphertext (since we want errorless communication).

Notice, Eve does not know $S_a$ or $S_b$, even though $S_a$ and $S_b$ could have a common piece of *shared secret* $s$. This $s$ is not known to Eve, but is known to both Alice and Bob.

## 2 INITIAL ASSUMPTIONS

To describe the nature of $E$ and $D$ more precisely, some assumptions about the agents must first be made.

- The encryption and decryption algorithms are public information known by all agents.
- Alice must *always* be able to recover the correct message.
- Eve is computationally strong, i.e. has an unbounded computational power. This assumption will be relaxed later to allow for more realistic modeling, but we assume it for now.

## 3 OBSERVATIONS

Some observations follow from these assumptions:

1. **Alice must have a secret from Eve.**

   *Rationale:* Alice must be able to decrypt the message from Bob. If Eve knows everything Alice knows, then Eve is functionally equivalent to Alice. Namely, if Alice can read the message then so can Eve. This contradicts the problem definition.

2. **Bob must have a secret from Eve. In fact, Alice and Bob must share a secret $s$ not known to Eve.**

   *Rationale:* Let Bob's message be $m$, and let $X$ be all the other information he uses to produce $c$ (includes his randomness, possible secret key, etc.). Eve can go through all candidate values $m'$ for $m$ and $X'$ for $X$, until $E(m', X') = c$. When this happens, Eve is sure that $m = m'$ unless Alice knows a part of $X$. Indeed, otherwise and if $m$ is different from $m'$, from Alice's point of view the message could be both $m$ and $m'$, which contradicts unique decodability. Thus, the only way Eve should still be unsure if $m = m'$ is if Alice knows a part of $X$, i.e. Alice and Bob share a non-empty secret $s$ not known to Eve.

## 4 ONE TIME PAD

Now we have a pattern for encryption where Bob and Alice must have a shared secret $s$. We next describe a specific implementation of such an encryption system, called the *One Time Pad* (OTP). (Note that OTP is a specific scheme for achieving our goal but not the only such scheme possible.) In One Time Pad, we assume that the message $m$ is somehow chosen from the domain $\mathcal{M} = \{0,1\}^k$, while a shared secret key $s$ is chosen *at random* from the domain $\mathcal{S} = \{0,1\}^k$. Notice, in OTP Alice and Bob do not need any other secret information (beside the shared $s$ which is secret from Eve).

DEFINITION 1 The One Time Pad (OTP) encryption function is easily described; simply take the exclusive OR of the message string and the key $s$. This produces the cipher $c$, which can be decrypted by XORing it with the key $s$:

$$\begin{aligned} E(m,s) &= m \oplus s \\ D(c,s) &= c \oplus s \end{aligned}$$

$\diamondsuit$

Intuitively, OTP is good because every message $m \in \mathcal{M}$ and ciphertext $c \in \mathcal{C}$ correspond to the unique key $s = m \oplus c$. So without the key, all messages are indistinguishable to Eve.

**Remark 1** *Before the OTP, people had many other cryptosystem, including Ceasar cipher, shift cipher, substrution cipher, hill cipher, vigenere cipher, permutation cipher, and many others. Most of these ciphers are "insecure" (whatever this means for each of them). It is somewhat entertaining to study these ciphers and see why each of them is not secure. However, since this will not be useful for the remainder of the course, we will not do this here. Interested readers are referred to Chapter 1 of the Stinson's book.*

## 5 DEFINING PERFECT SECURITY

This seems like a good system, but we'd like stronger verification. What do we mean when we say that a system is secure? By "system" we don;t just mean the OTP system, but any encryption where Alice and Bob (necessarily) share some secret key. Since in our assumptions the adversary (Eve) can access the encrypted text, an intuitive definition of perfect security is this: A method is secure iff the "odds" of the adversary to figure out $m$ are the same whether or not he has $c$, i.e. seeing $c$ does not increase these "odds". Of course, the problem is in formalizing the above informal definition. To do this, assume a message $M$ was chosen by Bob from some probability distribution over $\mathcal{M}$ and Bob publicly announced this distribution. As we will see, the the definition below makes sense for any message distribution $M$, but for simplicity — and because it will suffice for our purposes — the reader may assume that $M$ is chosen by Bob uniformly at random from $\mathcal{M}$. At this stage, from Eve's point of view each possibility $M = m$ is equally likely. Now assume Bob computes the ciphertext $C = E_s(M)$ and gives the particular outcome (which happens to be some $c \in \mathcal{C}$) to Eve. We now estimate again the probability that $M = m$ *after Eve has learned that* $C = c$. If the system is ideally secure, this probability should not change (i.e. remain uniform over $\mathcal{M}$), irrespective of specific $m$ and $c$. Notice, the above probability is also taken over the random choice of the shared key $s$. Formally:

DEFINITION 2 Let $M \in \mathcal{M}$ be a random message and $C \in \mathcal{C}$ be the ciphertext of $M$, that is, $C = E_s(M)$. For any $m \in \mathcal{M}$ and $c \in \mathcal{C}$, an encryption system is called **perfectly secure** if from the perspective of the attacker, $Pr(M = m \mid C = c) = Pr(M = m)$. This means that Eve's probability of guessing $M$ remains unchanged after seeing any particular outcome $C = c$. $\diamondsuit$

We can now formally show that OTP is secure:

**Proof:** Take any $m \in \mathcal{M}$ and $c \in \mathcal{C}$.

$$\begin{aligned} Pr(M = m \mid C = c) &= \frac{Pr(M = m \wedge C = c)}{Pr(C = c)} \\ &= \frac{Pr(M = m) \cdot Pr(C = c \mid M = m)}{Pr(C = c)} \\ &= \frac{Pr(M = m) \cdot Pr(C = c \mid M = m)}{\sum_{\tilde{m} \in M} (Pr(M = \tilde{m}) \cdot Pr(C = c \mid M = \tilde{m}))} \end{aligned}$$

We obtain step 1 from Bayes's law. Step 2 comes from the definition of conditional probability. In step 3, we expand out $Pr(C = c)$, which is the sum of all the conditional probabilities that $Pr(C = c)$, weighted by the probability of the condition. Namely, we sum over all possible $\tilde{m}$'s.

We then note that $Pr(C = c \mid M = \tilde{m}) = Pr(s = c \oplus \tilde{m})$, as $\tilde{m} \oplus s = c$ iff message $\tilde{m}$ and unique key $s$ produce ciphertext $c$. Since every $s \in \{0,1\}^k$ is just as likely, the probability is uniform over the key space: $Pr(s = c \oplus \tilde{m}) = \frac{1}{2^k}$. Hence,

$$\begin{aligned} Pr(M = m \mid C = c) &= \frac{Pr(M = m) \cdot \frac{1}{2^k}}{\sum_{\tilde{m} \in M} \left( Pr(M = \tilde{m}) \cdot \frac{1}{2^k} \right)} \\ &= \frac{Pr(M = m)}{\sum_{\tilde{m} \in M} Pr(M = \tilde{m})} \\ &= \frac{Pr(M = m)}{1} \end{aligned}$$

Thus, $Pr(M = m \mid C = c) = Pr(M = m)$, so OTP's security has been proven. □

**Remark 2** *As we can see, the fact that $M$ is uniformly random was not used anywhere in the proof. Thus, OTP is perfectly secure for* any *message distribution $M$.*

## 6 PROBLEMS WITH ONE TIME PAD

Even though the OTP method is secure, it has several undesirable features. First, each message must use a new key. Indeed, if Bob sends using the same key $s$ two ciphertexts, $c_0 = m_0 \oplus s$ and $c_1 = m_1 \oplus s$, then we have $c_0 \oplus c_1 = (m_0 \oplus s) \oplus (m_1 \oplus s) = m_0 \oplus m_1$. Thus, Eve certainly learns at least $m_0 \oplus m_1$, which is a lot of information! In particular, if $m_0$ is later revealed to Eve (say, it is no longer important), Eve will learn $m_1$, which could still be important. Clearly, this should not happen in the the ideal system.

Thus, if OTP is to remain secure with multiple message, it seems like a new key has to be used per each message. But since the length of each key in OTP is equal to the length of the encrypted message, it means that the overall length of the shared secret must be equal to the total number of secretly communicated bits. This seems to be really prohibitive for long term communication. The immediate question that comes to mind is whether this negative phenomenon is a specific feature of the OTP (and maybe we could do better with a better scheme), or there is some deeper reason for this inefficiency that will prevail in *any* perfectly secure system. Unfortunately, the latter is the case as was shown by Shannon.

**Theorem 1 (Shannon)** *For* **any** *perfectly secure scheme where Alice and Bob share a key $s$ from space $\mathcal{S}$ and can encrypt any message $m$ from space $\mathcal{M}$, we must have $|\mathcal{S}| \geq |\mathcal{M}|$. Thus, OPT is optimal in this regard.*

**Proof:** Take any valid ciphertext $c$ (which, say, could correspond to some message $m_0$ under the appropriate choice of the key). Let us count the number of messages $m$ that could result from the decryption of $c$ under some valid secret key $s$. Call this number $A$ and let us estimate $A$ in two different ways. On the one hand, each key $s$ in $\mathcal{S}$ can correspond to at most one $m$, since Alice should decrypt $c$ in at most one way for each choice of $s$ (else unique decodability is gone).

Thus, $A \leq |\mathcal{S}|$. On the other hand, we claim that $A = |\mathcal{M}|$, i.e. *every* $m \in \mathcal{M}$ can result in producing $c$ when encrypted by Bob. Indeed, if this was not the case for some $m$, then the a-priori $\Pr(M = m) > 0$, while if $C = c$ (which could happen with non-zero probability, say if $M$ was equal to $m_0$ that could in turn result in $C = c$ by assumption), then $\Pr(M = m \mid C = c) = 0$, which would contradict the perfect security. Thus, $A = |\mathcal{M}| \leq |\mathcal{S}|$, completing the proof. □

The above result shows some severe limitations of perfect security: the key must be as long as the message. Hence, the problem of *key exchange* (i.e., Alice and Bob agreeing in advance of the key) becomes a real issue. Clearly, we would like to find a better, more practical way to encrypt our data.

## 7 COMPUTATIONALLY BOUNDED ADVERSARIES

These previous negative results do not mean that encryption has no hope of succeeding. Indeed, the assumption that Eve has unbounded computing power is perhaps too strong. In reality, Eve does not, so we make the reasonable assumption on Eve's computing power. In turns out, the realistic and convenient way to model this is to say that Eve has only *probabilistic polynomial time* (PPT) computing power. To be fair, we shall restrict Alice and Bob to PPT as well.

Before defining this concept, let us by first defining a *Polynomial Time Algorithm*.

DEFINITION 3 [poly-time (Polynomial Time) Algorithm] If an algorithm $A$ gets an input of size $k$, it is considered polynomial time if it runs in $O(k^c)$ time, where $c$ is a constant. We write $y = A(x)$ to denote the output of $A$ on input $x$. ♢

Now, let us define *probabilistic polynomial time* (PPT).

DEFINITION 4 [PPT (Probabilistic Polynomial Time) Algorithm] It is a polynomial time algorithm $A$ that is *randomized*. Namely, it is allowed to flip coins during its computation. We write $y = A(x;\ r)$ to denote the output of $A$ on input $x$, when $r$ were the internal coin tosses made by $A$.[1] We write $y \leftarrow A(x)$ to denote the *random variable* $y$ which corresponds to the randomized output of $A$ on input $x$. This means that $r$ was chosen at random and $y = A(x;\ r)$ was computed. ♢

Finally, only bounding the running time of Eve will not be sufficient to get out of the Shannon's impossibility result. We will make it a bit more formal later, but, intuitively, this is because Eve can always try to guess the value which is hard to compute, and this way get a "negligible" chance to break the resulting cryptosystem, even though it would take Eve a very long time to break it with any "significant" probability. In other words, although the "advatange" of Eve over an ideal system might be non-zero, it will be so small (as long as Eve is PPT), that for all effective purposes it can simply be ignored.

This leads to the following definition of a *negligible function*.

DEFINITION 5 [Negligible Function (negl$(k)$)] A function $v(k)$ is called negligible, and denoted negl$(k)$, if:

$$(\forall c > 0)\ (\exists k')\ (\forall k \geq k')\quad \left[v(k) \leq \frac{1}{k^c}\right]$$

♢

[1] Notice, the length of $r$ must necessarily be polynomial in $k$.

In other words, we will use the notation $\mathsf{negl}(k)$ to say that some function of interest $v(k)$ (typically, the probability of Eve "breaking the system on inputs of size $k$") is less than the inverse of any polynomial for sufficiently large $k$.

The reason for this choice is because negligible probability of success stays negligible after even a polynomially many attempts to break the system: notationally, $\mathsf{poly}(k) \cdot \mathsf{negl}(k) = \mathsf{negl}(k)$.

WHAT IS $k$? In algorithms, $k$ usually denotes the "size of the problem", which is natural for the problem at hand. E.g., for sorting it is the size of the array to be sorted, for graph algortihms it is the number of vertices or edges of the graph, and so on. In cryptography, most tasks usually have many participants who take different inputs. For example, in our scenario we already had Alice, Bob, and Eve, who all had different inputs. Despite that, for most cryptographic problems it is also natural to define "the size of the problem", which in cryptography is usually called the *security parameter*. After the meaning of this parameter $k$ is specified, technically, every value of $k$ defines a different and "larger" scenario. For example, for the one-time pad cryptosystem we chose $k$ to denote the message size. Then, different $k$'s correspond to different and "larger" cryptosystems encryptinglonger and longer messages.

More generally, in more complex situations, once natural $k$ is chosen as a security parameter, we insist that all the honest parties (like Alice and Bob) run in some *fixed* (probabilistic) polynomial time in $k$ (e.g., for the OTP case Alice and Bob where linear in $k$). On the other hand, Eve can run in $\mathsf{PPT}$ in $k$, and should have at most negligible probability of breaking the system, as a function of $k$. This is why $k$ is called the "Security parameter", since by increasing $k$ we get higher and higher security (although also slightly degraded efficiency for "honest" users). Thus, there is a tradeoff in determining the "correct" value of $k$: we do not want to make it low, so that the system is secure, but also do not want to make it unnecessarily large, so that we do not waste the resources of honest parties if the system is already "secure" for a smaller value of $k$.

So how should once choose $k$ in practice? Well, after analyzing a given system, we hope that we can upper bound the advantage $\varepsilon$ of Eve as a function of Eve's running time $t$ (say, $2^{60}$) and the security parameter $k$. In a good cryptosystem, this function is negligible in $k$ (e.g., perhaps decaying like $2^{-k}$). Thus, for a relatively small $k$, Eve's advantage $\varepsilon$ will be smaller than some "acceptable" and "unnoticeable" threshold (say, $2^{-60}$). And this is precisely the value $k$ one chooses as a security parameter in practice. If later this $k$ is too small, one increases $k$ to make the system more secure (at the expense of having Alice and Bob run a bit slower). Ideally, though, one leaves enough "slack" in the initial choice $k$, so that reasonable increases in computer power still leave the system secure for decades to come!

In any event, at this stage this is a bit abstract, and we will come back to this point later, but we summarize our discussion as follows:

- One usually does not build a single cryptosystem, but parametrizes one's cryptosystem by a conveniently chosen security parameter $k$. Then "efficient" means "polynomial in $k$", and "negligible" means "negligible function of $k$".

- Intuitively, concrete $k$ is later chosen in practice such that no attacker running in time "significantly less" than $2^k$ has advantage significantly greater than $2^{-k}$. In a good system, this $k$ will still be small enough, so Alice and Bob, who run in time poynomial in $k$ (say, $O(k^2)$), still run pretty fast.

**Remark 3** *Finally, we mention a trivial notional convention which we will repeatedly use without further explanation from now on. As we said, all participants are allowed to run in time polynomial in $k$. But how do they "know" $k$? Technically, we will have to explicitly supply $k$ as part of the input. However, instead of giving them $k$ in binary, which is only $\log k$ bits long, we will give $k$ in unary, which is indeed $k$ bits long. This is done to ensure that all our algortihms — which are polynomial in their* input length *— are allowed to run in time polynomial in the security parameter $k$. Thus, providing $k$ in unary is a convenient way to ensure that polynomial in $k$ time is allowed. Notationanly, $k$ in unary is denoted $1^k$. Thus, when an algortihm takes this "mysterious" $1^k$, this is simply to explicily tell it that the security parameter is equal to $k$.*

## 8 FINDING A BETTER SYSTEM

In light of these new relaxed assumptions on Eve, we must re-examine the observations we made earlier.

1. **Alice has a secret from Eve.**

   Clearly this observation must still hold, for the same reason given initially. Indeed, since Alice is now PPT, Eve still has enough power to decrypt the message if the assumption was false.

2. Bob has a secret from Eve. In fact, Alice and Bob must share a secret not known to Eve.

   This observation now does not seem to be necessarily true: if Eve is not strong, then she can no longer resort to brute force methods to reverse-engineer Bob's encrypted message. Indeed, we will give strong evidence that this observation is false.

For now, though, we define two important cryptographic models, depending on whether or not we assume the (no longer required) Observation 2 above. If we do, we come back to the same problem of secure communication with the shared key. This encryption setting is called *Symmetric-Key (or Private-Key)* Encryption (SKE). Since OTP is still a valid solution, but the proof of Shannon's theorem no longer holds when Eve is computationally bounded, the main goal of SKE is to **achieve "secure communication" with the secret key *much shorter* than the length of the messages exchanged**. We will come back to this challenging problem, but for now we discuss the new model, where we no longer assume that Alice and Bob share the secret (i.e., Observation 2 is false).

## 9 NOT SHARING A SECRET (PUBLIC-KEY ENCRYPTION)

In fact, we will assume that *Eve knows everything that Bob does.* From Observation 1, Alice should still have a secret $S_a$ (which we now denote by $SK$, to stand for the "secret key") from Eve (and, hence, from Bob as well). We also allow for some public information $PK$ (stands for "public key") available to all the parties. Notice, since Eve knows what Bob does, Eve, and in fact *anyone else*, can encrypt messages for Alice. However, we want *only Alice to be able to understand them.* This encryption setting is called *Public-Key (or Asymmetric-Key)* Encryption (PKE). Since the PKE model was impossible when Eve was unbounded, the main goal of PKE is **to achieve it at all** (only later concentrating on the secondary efficiency issues).

We now briefly compare the the SKE and the PKE. In SKE, at least an inefficient solution (OTP) clearly exist, but the problem is that of secure key-exchange. In PKE, the solution could not exist in the unbounded setting. However, if we are to find a solution, it will not have the key exchange problem. Indeed, since *anyone* can encrypt messages for Alice using her known public key $PK$, Alice simply needs to publish her $PK$ once, and does not need to meet each possible recipient. On the other hand, the communication is *asymmetric*: in SKE both Alice and Bob could send the messages to each other using the shared key, while in PKE all the messages are intended for Alice, i.e. a new PKE structure is need for Bob to get messages. Finally, we will later see that in practice, the solutions to SKE seem to be much more efficient than those for PKE. Thus, PKE adds convenience at the cost of efficiency. More on this later in the course.

Since it seemed more interesting to do something that was not possible before at all, rather than break the "Shannon barrier" for something which was principally possible, historically people attacked the PKE first (even though we will later see that SKE is an *easier* problem in some sense). We will follow the same historic approach for now.

## 10 TOWARDS PKE... TO ONE-WAY FUNCTIONS

We start by making some specific assumptions to simplify the problem as much as we can. We start by assuming that the encryption algorithm $E$ is not probabilistic. As we'll see soon, this assumption is *wrong*, but it would be a good simplification for now. Also, since the public key $PK$ is fixed once originally chosen, we we define a function $f(m) = E_{PK}(m)$. The question is what properties should such a function $f$ satisfy to yield a "reasonably good" PKE? Going though our desiderata, we claim that $f$ should be:

1. **Invertible:** It must be possible for Alice to decrypt encrypted messages.
2. **Efficient to compute:** It must be reasonable for people to encrypt messages for Alice.
3. **Difficult to invert:** Eve should not be able to compute $m$ from the "encryption" $f(m)$.
4. **Easy to invert given some auxiliary information:** Alice should restore $m$ using $SK$.

It turns out that properties 2 and 3 (easy to compute but difficult to invert) capture the main difficulty in designing such an $f$. Therefore, functions satisfying just properties 2 and 3 alone received some special treatment. Such functions are called *one-way functions* (OWF's). Adding property 1 on top will yield the notion of *one-way permutations* (OWP's), while finally throwing property 4 gives *trapdoor permutations* (TDP's).

For syntactic convenience, in the definitions below the input $x$ will replace the "message" $m$ and the trapdoor $T$ — the "secret key" $SK$. Also, $\mathsf{poly}(k)$ denote some polynomial function in $k$, whose particular form is unimportant (as long as it exists).

DEFINITION 6 A function $f(x)$ is a *One-Way Function* iff there exists a polynomial time algorithm to compute $f(x)$ and for any PPT algorithm $A$ (this is the adversary's algorithm) trying to "invert" $f$, we have

$$P(f(z) = f(x) \mid x \in \{0,1\}^k, y \leftarrow f(x), z \leftarrow A(y, 1^k)) = \mathsf{negl}(k)$$

Namely, any PPT $A$ succeeds with only a negligible probability in in finding *any* valid preimage $z$ of $y = f(x)$. In case when $f$ is also one-to-one, $f$ is called a *One-Time Permutation*. $\diamondsuit$

We remark that in the definition of the OWF we had to require that $A$ succeeds if $f(x) = f(z)$, rather than the "natural" $x = z$. This is because we did not require $f$ to be a permutation yet. Indeed, if we changed $f(x) = f(z)$ to $x = z$, the trivial function $f(x) \equiv 0$ will trivially satisfy the definition (why??). Of course, this problem does not arise for OWP's, since there the condition $f(x) = f(z)$ is clearly equivalent to $x = z$ (i.e. $x$ is only preimage of $y = f(x)$).

DEFINITION 7 A *Trapdoor Permutation* $f$ is a OWP with the extra property that for every $n$, there exists a string $T$ (the "trapdoor") of polynomial length, $|T| \leq \mathsf{poly}(k)$, and a polynomial-time "inversion" algorithm $I$ such that $I(f(x), T) = x$, for any $x \in \{0,1\}^k$. $\diamondsuit$

Notice, that since inverting $f$ is hard without anything and easy with the trapdoor, it means that it *must* be hard to compute the trapdoor (i.e., find the "secret key"). Notice, the existence of the trapdoor seems like the essence for constructing PKE: everyone has access to the encrypting function, but the only person possessing the "auxiliary information" has the ability to invert $E_{PK}(x)$ and thus decrypt encrypted messages.

Next lecture we will see the specific number-theoretic examples of the OWF's, OWP's and TDP's, as well as try to see if they indeed suffice for solving our problem of secure communication.

**Remark 4** *Mathematically speaking,* OWF*s,* OWP*s and* TDP*s should also include a* PPT *key generation algorithm* Gen*, which, on input* $1^k$*, outputs the public key* $PK$ *desribing* $f$*, and, in case of* TDP*s, also the corresponding trapdoor information* $T$*. This way we know how to practically sample a corresponding hard-to-invert function. Also, although we will assume for now that the message space of* $f$ *is* $\{0,1\}^k$*, in general the domain of* $f$*, denoted* $\mathcal{M}_k$*, could also be generated as part of the key generated algortihm* Gen*, as long as one can efficiently sample a random element* $x$ *from* $\mathcal{M}_k$*. We will come back to this point later.*

| CSCI-GA.3210-001<br>MATH-GA.2170-001 | **Introduction to Cryptography** February 1, 2012 |
|---|---|

# Lecture 2

*Lecturer: Yevgeniy Dodis* *Spring 2012*

This lecture begins with a discussion on secret-key and public-key cryptography, and then discusses One-way Functions (OWF), and their importance in cryptography. Essentially, an OWF is easy to compute, but difficult to invert. A One-way Permutation (OWP) is an OWF that permutes elements from a set. A Trapdoor Permutation (TDP) is essentially an OWP with certain secret information, that if disclosed, allows the function to be easily inverted.

No OWF is known to exist unconditionally, since showing the existence of OWFs is at least as hard as showing that complexity class P is different from NP — a long standing open problem. However, there exists several good candidates for OWF, OWP, and TDP. We provide such example later, but start with sample applications of general OWFs, OWPs and TDPs. First, we will see how the assumption of the existence of OWF leads to a secure password-authentication system. Next, we show that a widely used S/Key Password Authentication System is secure using any OWP (but not general OWFs). Finally, we deascribe a (very weak) public-key encryption scheme based on TDPs.

We also briefly provide (conjectured) number-theoretic candidates of a OWF, a OWP and a TDP: Prime Product as an example of a OWF candidate, Modular Exponentiation as an example of a OWP candidate, and RSA as an example of a TDP candidate. More detailed introduction to number theory will be given next lecture.

Finally, we describe some criticisms regarding OWFs, OWPs, and TDPs in practical applications, and give suggestions of how to overcome these criticisms.

## 1 Symmetric and Public-Key Encryption

We briefly recap these notions below.

1. **Secret-Key (Symmetric-Key) Encryption**

   - **Before the Encryption**
     Bob and Alice have some sort of secret key $S$ they arranged to use in advance that Eve does not know.
   - **Encryption**
     When Bob wishes to send Alice a plaintext message $M$ via the Internet, Bob encrypts $M$ using secret key $S$ to form a ciphertext $C$. (Formally, we summarize encryption with $S$ as $E_S$, and say that $C = E_S(M)$.) Bob then sends $C$ over the Internet to Alice.
   - **Decryption**

Upon receiving $C$, Alice uses the same secret key $S$ to decrypt $C$, giving her $M$, the original plaintext message. (Formally, we summarize decryption with $S$ as $D_S$ and say that $D_S(C) = D_S(E_S(M)) = M$.)

- **Eve's Standpoint**
  Eve only sees $C$ being sent over the Internet. She has no knowledge of $S$. Very informally (stay tuned!), the scheme is "Secure" if it is hard for Eve to learn about $S$ or plaintexts based on the ciphertexts.

A more formal definition of symmetric-key encryption is given below (only syntax, not security yet!):

DEFINITION 1 [Symmetric-key encryption ($\mathsf{SKE}$)] A $\mathsf{SKE}$ is a triple of $\mathsf{PPT}$ algorithms $\mathcal{E} = (G, E, D)$ where:

(a) $G$ is the *key-generation algorithm*. $G(1^k)$ outputs $(S, \mathcal{M})$, where $S$ is the shared secret key, and $\mathcal{M} = \mathcal{M}(S)$ is the (compact description of the) message space associated with the scheme. Here $k$ is an integer usually called the *security parameter*, which determines the security level we are seeking for (i.e., everybody is polynomial in $k$ and adversary's "advantage" should be negligible in $k$). Very often, $S$ is simply a random string of length $k$, and $\mathcal{M}$ is $\{0,1\}^*$.

(b) $E$ is the *encryption algorithm*. For any $m \in \mathcal{M}$, $E$ outputs $c \xleftarrow{r} E_S(m, S)$ — the encryption of $m$. $c$ is called the *ciphertext*. We usually write $E(m, S)$ as $E_S(m)$ (or $E_S(m;\ r)$, when $E$ also takes randomness $r$ and we wish to emphasize this fact).

(c) $D$ is the (deterministic) *decryption algorithm*. $D(c, S)$ outputs a value $\tilde{m} \in \{\mathsf{invalid}\} \cup \mathcal{M}$, called the decrypted message. We also usually denote $D(c, S)$ as $D_S(c)$.

(d) We require the **correctness** property:

$$\forall m \in \mathcal{M}, \quad \tilde{m} = m, \quad \text{that is} \quad D_S(E_S(m)) = m$$

$\diamondsuit$

2. **Public-Key Encryption**

- **Before the Encryption**
  Alice publishes to the world her public key $PK$. Therefore, both Bob and Eve know what $PK$ is. This public key is only used to encrypt messages, and a separate key $SK$ is used to decrypt messages. (This is unlike the Secret-Key scheme where one key $S$ is used to both encrypt and decrypt.) Only Alice knows what $SK$ is, and nobody else, not even Bob.
- **Encryption**
  When Bob wishes to send Alice a plaintext message $M$ via the Internet, Bob encrypts $M$ using Alice's public key $PK$ to form a ciphertext $C$. (Formally, we summarize encryption with $PK$ as $E_{PK}$ and say that $C = E_{PK}(M)$.) Bob then sends $C$ over the Internet to Alice.

- **Decryption**
  Upon receiving $C$, Alice uses her secret private key $SK$ to decrypt $C$, giving her $M$, the original plaintext message. (Formally, we summarize decryption with $SK$ as $D_{SK}$ and say that $D_{SK}(C) = D_{SK}(E_{PK}(M)) = M$.)
- **Eve's Standpoint**
  Unlike the Secret-Key scheme, Eve knows everything Bob knows and can send the same messages Bob can. And, only Alice can decrypt. And, when Bob sends his message, Eve only sees $C$, and knows $PK$ in advance. But, she has no knowledge of $SK$. Very informally (stay tuned!), the system is "secure" if it is hard for Eve to learn about $SK$ or plaintexts based on ciphertexts and $PK$.

A more formal definition of public-key encryption is given below (only syntax, not security yet!):

DEFINITION 2 [Public-key encryption (PKE)] A PKE is a triple of PPT algorithms $\mathcal{E} = (G, E, D)$ where:

(a) $G$ is the *key-generation algorithm*. $G(1^k)$ outputs $(PK, SK, \mathcal{M})$, where $SK$ is the secret key, $PK$ is the public-key, and $\mathcal{M}$ is the (compact description of the) message space associated with the $PK/SK$-pair. As before, $k$ is an integer the *security parameter*.

(b) $E$ is the *encryption algorithm*. For any $m \in \mathcal{M}$, $E$ outputs $c \xleftarrow{r} E(m,\ PK)$ — the encryption of $m$. $c$ is called the *ciphertext*. We sometimes also write $E(m, PK)$ as $E_{PK}(m)$ (or $E_{PK}(m;\ r)$, when we want to emphasize the randomness $r$ used by $E$).

(c) $D$ is the (determinstic) *decryption algorithm*. $D(c,\ SK)$ outputs $\tilde{m} \in \{\mathsf{invalid}\} \cup \mathcal{M}$, called the decrypted message. We also sometimes denote $D(c,\ SK)$ as $D_{SK}(c)$.

(d) We require the **correctness** property:

$$\forall m \in M, \quad \tilde{m} = m, \quad \text{that is} \quad D_{SK}(E_{PK}(m)) = m$$

♢

## 2 Primitives

We mentioned the following three primitives commonly used in Cryptography:

1. OWF: One-Way Functions
2. OWP: One-Way Permutations
3. TDP: Trap-Door Permutations

The next sections will define these primitives and give conjectured candidates of each one (we say "candidate", and not "example", because the formal existence of OWF, and consequently OWP and TDP, has yet to be proven).

## 3 OWF

A function is One-way if it is easy to compute, but difficult to invert. More formally,

DEFINITION 3 [OWF] A function $f : \{0,1\}^* \to \{0,1\}^*$ (or, a function $f$ that maps finite strings to finte strings) is *one-way* it is satisfies two properties:

1. $\exists$ poly-time algorithm which computes $f(x)$ correctly $\forall x$. (Thus, easy to compute.)
2. $\forall$ PPT Algorithm $A$,

$$\Pr(f(z) = y \mid x \leftarrow^R \{0,1\}^k;\ y = f(x);\ z \leftarrow^R A(y, 1^k)) \leq \mathsf{negl}(k)$$

where $\leftarrow^R$ means randomly chosen. (So $x$ is randomly chosen from the set of $k$-bit numbers, and $z$ is randomly outputted from algorithm $A$ when it has $y$ as input.) Thus, $f$ is hard to invert. So, in polynomial time (in $k$) Eve has probability $\mathsf{negl}(k)$ or less of figuring out *any* preimage of $f(x)$.

$\diamondsuit$

And, keep in mind that no proof derived yet shows that OWF's exist. However, there's good evidence OWF's do exist.[1] And later on, candidate OWF's will be shown.

COLLECTION OF OWFS. The above definition of a OWF is very convenient to us when building other objects from OWFs. (In particular, we will usually use it in the homework and exams.) However, it is somewhat inconvenient to use if we want to *build* OWFs in practice. For one thing, the domain of a OWF is $\{0,1\}^*$, which is not the case for practical constructions. Also, the above definition does not allow one to randomly choose and publish some public parameters which woiuld actually define a given OWF $f$. Finally, we see that this definition will be hard to extend to *trapdoor permutations*, which we will do soon. Therefore, in order to include practical constructions, we define a collection of OWFs (which still suffice for all the applications of OWFs) as follows.

DEFINITION 4 [colelction of OWF] A *collection of one-way functions* is given be three PPT algorithms (Gen, Eval, Sample).

1. Gen is the *key-generation algorithm*. $G(1^k)$ outputs a public key $PK$, which implicitly defines a domain $D = D(PK)$ and the range $R = R(PK)$ of a given one-way function $f_{PK}$ (which we simply denote by $f$ when $PK$ is clear from the context). As before, $k$ is an integer the security parameter.
2. Eval is the (deterministic) *function evaluation algorithm*. $\mathsf{Eval}(x, PK)$ computes the value $f_{PK}(x) \in R$, for any $x \in D$. We usually write $f_{PK}(x)$, or just $f(x)$, to denote the output of $\mathsf{Eval}(x, PK)$.
3. Sample is a probabilistic algortihm, which, on input $PK$, samples a random element from the domain $D$ of $f_{PK}$. We write $x \leftarrow D(PK)$, or just $x \leftarrow D$, as a shorthand for running $\mathsf{Sample}(PK)$.

[1] Those familiar with the Complexity Theory can observe that the existence of OWFs trivially implies that $P \neq NP$, which is a long-standing open problem. In fact, we do not know how to prove OWFs exist even if we assume that $P \neq NP$!

With this notation, the security of a collection of OWFs is very similar to security of OWFs defined earlier: $\forall$ PPT Algorithm $A$,

$$\Pr(f_{PK}(z) = y \mid PK \leftarrow \mathsf{Gen}(1^k);\ x \leftarrow D(PK);\ y = f_{PK}(x);\ z \leftarrow A(y, PK, 1^k)) \leq \mathsf{negl}(k)$$

$\diamondsuit$

As we can see, collection of OWFs allow us a bit more freedom in specifying the domain and range of $f$, as well as publishing some parameter $PK$ describing $f$. We will see examples later on.

# 4 OWP

A function $f$ is a OWP if it is OWF, and a permutation. More formally,

DEFINITION 5 [OWP] A function $f$ is OWP if:

1. It satisfies all requirements for being OWF.
2. It is a permutation (that is every $y$ has a unique preimage $x$).

$\diamondsuit$

Similarly, one can define *collection of* OWP*s* in the same manner as we defined collection of OWFs. The only addition is that $D(PK) = R(PK)$, and $f_{PK}$ must be a permutation (be one-to-one and onto) over $D(PK)$.

# 5 TDP

A function $f$ is a TDP if it is OWP, and given certain information, $f$ can be inverted in PPT. More formally, one can attempt to define TDPs similar to Definition 3 and Definition 5 as follows. A function $f$ is TDP if (a) it satisfies the requirements for OWP from Definition 5; and (b) There exists a poly-time algorithm $I$, some constant $c$, and a string $t_k$ (for each $k$) such that, for all large enough $k$, the size of $t_k$ is at most $O(k^c)$, and for any $x \in \{0,1\}^k$, $I(f(x), t_k) = z$ where $f(z) = f(x)$.

A moment relection, however, shows that this definition seems to be quite useless in applications. Indeed, one may wonder what is the usage of TDPs, if nobody can find the trapdoor information $t_k$ in polynomial time (if one could, then one would be able to invert it, and contradict the one-wayness of $f$). Indeed, this observation is correct, and this is why TDPs are only defined as a *collection* of functions, similarly to a collection of OWFs and OWPs we defined earlier. Indeed, such collections have a key generation algortihm Gen which outputs the public key $SK$. Now, we simply let Gen also output the trapdoor information (which we denote $SK$) which would allow one to always invert a given TDP. More formally,

DEFINITION 6 [collection of TDP] A *collection of trapdoor permutations* is given be four PPT algorithms (Gen, Eval, Sample, Invert).

1. Gen is the *key-generation algorithm.* $G(1^k)$ outputs a public key $PK$, which implicitly defines a domain $D = D(PK)$ of a given function $f_{PK}$ (which we simply denote by $f$ when $PK$ is clear from the context), and a corresponding secret (or *trapdoor*) key $SK$.

2. Eval is the (deterministic) *function evaluation algorithm.* $\mathsf{Eval}(x, PK)$ computes the value $f_{PK}(x) \in R$, for any $x \in D$. We usually write $f_{PK}(x)$, or just $f(x)$, to denote the output of $\mathsf{Eval}(x, PK)$.

3. Sample is a probabilistic algortihm which, on input $PK$, samples a random element from the domain $D$ of $f_{PK}$. We write $x \leftarrow D(PK)$, or just $x \leftarrow D$, as a shorthand for running $\mathsf{Sample}(PK)$.

4. Invert is a (deterministic) inversion algortihm which, on input $y \in D$ and a secret key $SK$ computes a value $z \in D(PK)$ such that $f_{PK}(z) = y$. We usually write $z = f_{PK}^{-1}(y)$, or just $z = f^{-1}(y)$ to denote the output of $\mathsf{Invert}(y, SK)$, with an implicit understanding that $f^{-1}$ is only easy to compute with the knowledge of the trapdoor key $SK$.

The security of TDPs is idential to that of OWPs, where the attacker cannot invert $f$ without the trapdoor key $SK$: $\forall$ PPT Algorithm $A$,

$$\Pr(z = x \mid (PK, SK) \leftarrow \mathsf{Gen}(1^k);\ x \leftarrow D(PK);\ y = f_{PK}(x);\ z \leftarrow A(y, PK, 1^k)) \leq \mathsf{negl}(k)$$

$\diamondsuit$

# 6 Number-Theoretic Examples of OWF, OWP, TDP

BEFORE YOU BEGIN: **If you are not very familiar with number theory, I strongly recommend you skip this section on first reading.** *Indeed, the main purpose of this somewhat dense section is to make the treatment of* OWF*,* OWP*s and* TDP*s a bit less "abstract", by giving examples we will study in more detail later. In particular, if "abstract" is fine with you, you can move directly to the next "application" section. In other words, the specific examples are not needed to understand the remainder of this lecture. We will study number theory in more detail the next lecture.*

As we said, we do not know how to unconditionally prove the existence of OWFs, OWPs and TDPs. However, we have several plausible candidates. In this section we give a sample candidate for each primitive based on *number theory.*[2]

## 6.1 OWF Candidate: Integer Multiplication

Let's define a function $f$ as $f(p, q) = p * q$, where $p$ and $q$ are $k$-bit primes and $*$ is the regular integer multiplication. (So, our domain is the set of pairs of $k$-bit primes, and

[2]For OWPs and TDPs, all examples that we know use number theory. For OWFs, there are many other proposed candidates. However, we will study them a bit later in the course, concentrating on number-theoretic examples for now.

$f : \mathbb{P}_k * \mathbb{P}_k \to X$ where $\mathbb{P}_k$ is the set of $k$-bit primes, and $X$ is the set of $2 * k$-bit numbers whose factorization is made up of two $k$-bit primes.) Clearly, $f$ is not a permutation, but this is fine.

Let us denote $n = p * q$, so that $f(p, q) = p * q = n$. Upon seeing $n$, there's no known polynomial time algorithm $A$ such that $A(n)$ outputs values $p'$ and $q'$ so that $p' * q' = n$ (since we assume $n$ is a product of two primes, then either $p' = p, q' = q$, or $p' = q, q' = p$).

You might think, "Hey! That's not true! Just test all the numbers from 2 to $\sqrt{n}$." And propose a program that works like the following:

*for* $i = 2$ *to* $\sqrt{n}$ *do*
    *if (i divides n) then output(i,*$\frac{n}{i}$*);*

And you would then claim that your program runs in time $O(\sqrt{n})$, which is polynomial in terms of $n$. However, keep in mind that the number $n$ inputted into this algorithm is of magnitude roughly $2^{2k}$ and of length $2k$. Thus, since $\sqrt{n} \approx \sqrt{2^{2k}} \approx 2^k$, this algorithm runs in time $O(2^k)$, which is exponential in terms of the input size. Thus, this algorithm runs in exponential time. And, no algorithm that's polynomial (or even probabilistically polynomial) in terms of $k$ is known that can factor $n$.

Therefore, this function $f$ is easy to compute and *seems* difficult to invert, making it a good candidate OWF.

**Conjecture 1** *Integer multiplication is a* OWF.

This conjecture has stood up to attempts to disprove it for a long time, and forms one of the most common assumptions in cryptography. Usually, when we will use the above conjecture as an *assumption* to prove some result $X$, we will say

"If *factoring is hard*, then $X$ is *provably* true."

Of course, this does not mean we have proven $X$ unconditionally, but it means that the *only* way to break $X$ is to break factoring, and this seems quite unlikely given the long history of this problem!

## 6.2 OWP Candidate: Modular Exponentiation

We will define the function $f(x, p, g) = (g^x \bmod p, p, g)$, where the explanation of the above letters will follows shortly. Notice, however, that since $p$ and $g$ are "copied" through, for convenience instead of this $f$ we will write $f_{p,g}(x) = g^x \mod p$ (implicitly implying that $p$ and $g$ are part of the randomly generated input which are also part of the output). As we can see, in this example it is actually more convenient to use the notion of *collection* of OWPs. In this case, we can say that "copied" $p$ and $g$ are simply part of the randomly generated public key $PK$.

Let us now make some important math definitions, theorems, and observations to make the notation above clear. As stated, these can be skipped upon first reading (also see Lecture 3 for more extensive treatment), especially since several facts are stated without proof here.

MOMENTARY DETOUR. For any $n$, $\mathbb{Z}_n$ is the set of integers from 0 to $n - 1$. The multiplicative group of $\mathbb{Z}_n$, $\mathbb{Z}_n^*$, is defined as:

DEFINITION 7 $\mathbb{Z}_n^* = \{x \mid x \in \mathbb{Z}_n \wedge \gcd(x, n) = 1\}$ $\diamondsuit$

So, $\mathbb{Z}_n^*$ is the set of elements from $\mathbb{Z}_n$ that are relatively prime to $n$. Also note that for any $n$, $0 \notin \mathbb{Z}_n^*$. Also, note that for prime number $p$, $\mathbb{Z}_p^* = \mathbb{Z}_p - \{0\}$. This is because every number in $\mathbb{Z}_p$ except 0 is relatively prime to $p$.

Additionally, for any positive integer $n$, the set $\mathbb{Z}_n^*$ and the multiplication operator (we will often write $ab$ to denote $a * b \bmod n$) form a group. This is because:

1. $(\forall a, b \in \mathbb{Z}_n^*)[a * b \in \mathbb{Z}_n^*]$
2. $\mathbb{Z}_n^*$ has an identity element, which is 1. This is because $(\forall a \in \mathbb{Z}_n^*)[a * 1 = 1 * a = a]$
3. $(\forall a \in \mathbb{Z}_n^*)(\exists a' \in \mathbb{Z}_n^*)$ such that $a * a' = a' * a = 1$. (I.e., every element in $\mathbb{Z}_n^*$ has an inverse.) This is because for every $a \in \mathbb{Z}_n^*$, $\gcd(a, n) = 1$. Therefore, there exist integers $a', n'$ such that:

$$aa' + nn' = 1 \quad \Rightarrow \quad aa' = 1 + n(-n') \quad \Rightarrow \quad aa' \equiv 1 \mod n$$

(And, do keep in mind that this fact isn't always true if we were to deal with elements in $\mathbb{Z}_n$.)

Fermat's little theorem states that:

**Theorem 2 (Fermat's Little Theorem)** *For any prime $p$ and $x \in \mathbb{Z}_p^*$, $x^{p-1} = 1 \mod p$*

Also, for some arbitrary $a \in \mathbb{Z}_p^*$, the smallest $x$ where $a^x = 1 \mod p$ is referred to as the order of $a$ in $\mathbb{Z}_p^*$. (And, there may be elements in $\mathbb{Z}_p^*$ with order less than $p - 1$. For example, if $p$ is a prime larger than 3, then $(p-1)^2 = (-1)^2 = 1 \mod p$, so the order of $p - 1$ in $\mathbb{Z}_p^*$ is 2, and $2 < p - 1$ when $3 < p$)

And, also note the following number theory theorem.

**Theorem 3 (Number Theory Fact)** *When $p$ is prime, $\mathbb{Z}_p^*$ has at least one element $g$ with order $p - 1$.*

And, elements in $\mathbb{Z}_p^*$ with order $(p - 1)$ are commonly referred to as primitive elements or *generators*.[3] Notice that $\{g^1, g^2, ..., g^{p-1}\} = \mathbb{Z}_p^*$. So, raising $g$ to powers ranging from 1 to $p - 1$ (or 0 to $p - 2$, since $g^{p-1} = 1 = g^0 \mod p$) "generates" all the elements in $\mathbb{Z}_p^*$.

COMING BACK TO OWPS. As a result of all these condensed definitions, we can revisit our function $f_{p,g}$ where $f_{p,g}(x) = g^x \mod p$. Here $p$ is a $k$-bit prime, $g$ is a generator of the set $\mathbb{Z}_p^*$, and $x \in \mathbb{Z}_p - \{0\}$ is the actual input.

We now justify why we believe that $f_{p,g}$ is a OWP. First, since $g$ is a generator, our function could be viewed as a permutation from $\mathbb{Z}_p - \{0\} = \mathbb{Z}_p^*$ to $\mathbb{Z}_p^*$. Second, we claim that computing $y = g^x \bmod p$ could be done in polynomial time as follows. Assuming $p$ has $k$ bits, for any arbitrary $x \in \mathbb{Z}_p^*$, we can find $f_{p,g}(x)$ as follows:

[3] The origin of the term "primitive element" is puzzling. For example, it is certainly non-trivial to show that primitive elements exist in $\mathbb{Z}_p^*$, as stated by the above theorem.

1. Compute $g^{2^i} \mod p$ for every value of $i$ from 0 to $\log_2(p)$, which involves repeated squaring, and takes $\Theta(k)$ multiplications.

2. Look at the binary expansion of $x$, which might look like: $10011\ldots$, and note that $x = 2^{k-1}b_{k-1} + \ldots + 2^1 b_1 + 2^0 b_0$ where each $b_i$ represents a binary digit in $x$.

3. By plug-in, since

$$g^x = g^{2^{k-1}b_{k-1}+\ldots+2^1 b_1+2^0 b_0} = g^{2^{k-1}b_{k-1}} \cdots g^{2^1 b_1} g^{2^0 b_0} = (g^{2^{k-1}})^{b_{k-1}} \cdots (g^{2^1})^{b_1} (g^{2^0})^{b_0}$$

and each $g^{2^i} \mod p$ has already been found, and each $b_i$ is either 0 or 1, then each $(g^{2^i})^{b_i}$ term modulo $p$ has a known value (either $g^{2^i}$ or 1), and computing the product of all the $(g^{2^i})^{b_i}$ terms modulo $p$ to give $g^x$ is doable in $O(k)$ multiplications.

Overall, we get $O(k^3)$ algorithm. Therefore, $f_{p,g}(x)$ is easy to compute. The inversion problem corresponds to finding $x$ s.t. $g^x = y \bmod p$, when given $g, p, y$ as inputs. This is known as the Discrete-Log Problem which is believed to be very hard. Therefore, $f_{p,g}$ is believed to be hard to invert. Because of $f_{p,g}$ is easy to compute, believed to be hard to invert, and definitely is a permutation, $f_{p,g}$ makes a good candidate OWP. Unfortunately, there's no known trapdoor information that can make inverting $f$ easy (which would make it a TDP).

**Conjecture 4** *Modular exponentiation is a* OWP.

Similar to factoring, this conjecture has stood up to attempts to disprove it for a long time, and forms one of the most common assumptions in cryptography. Usually, when we will use the above conjecture as an *assumption* to prove some result $X$, we will say

"If *discrete log is hard*, then $X$ is *provably* true."

## 6.3 TDP Candidate: RSA

An RSA function is defined as $f(x, n, e) = x^e \bmod n$. As before, we write for convenience $f_{n,e}(x) = x^e \mod n$, where $n$ is the product of two primes $p$ and $q$, $x \in \mathbb{Z}_n^*$, $e \in \mathbb{Z}_{\varphi(n)}^*$. Now for a bit more number theory. (The fun never stops!)

DEFINITION 8 [Euler phi-function] For any positive integer $m$, $\varphi(m)$ is the number of positive integers less than $m$ that are relatively prime to $m$. ♢

As you might have guessed, for any positive integer $m$, the number of elements in the set $\mathbb{Z}_m^*$ is $\varphi(m)$. For any prime $p$, $\varphi(p) = p - 1$, since there are $p - 1$ positive integers less than $p$, and they're all relatively prime to $p$. Additionally, if $n = pq$ where $p$ and $q$ are primes, then, $\varphi(n) = (p-1) * (q-1) = n - (p + q - 1)$.

Now for Euler's Theorem, which is more general than Fermat's Little Theorem (mentioned earlier).

**Theorem 5** *For any positive integer $m$ and any $a \in \mathbb{Z}_m^*$, $a^{\varphi(m)} = 1 \bmod m$.*

Now note that for our RSA function $f$, we have $e \in \mathbb{Z}^*_{\varphi(n)}$. This is to make sure that $e$ is relatively prime to $\varphi(n)$, so that there exists a $d \in \mathbb{Z}^*_{\varphi(n)}$ such that $ed = 1 \bmod \varphi(n)$, so that for our $f_{n,e}(x) = x^e \mod n$, we can get $x$ back by doing

$$(x^e)^d = x^{ed} = x^{ed \bmod \varphi(n)} = x^1 = x \bmod n$$

And, assuming $c = f_{n,e}(x) = x^e \mod n$ for some value $x$, and $d$ is the inverse of $e$ modulo $\varphi(n)$ for the rest of this section, note that then since $x \in \mathbb{Z}^*_n$, and $\mathbb{Z}^*_n$ with the multiplication operator is a group (for reasons mentioned earlier), then $x$ raised to any power is also an element of $\mathbb{Z}^*_n$. Therefore, $c \in \mathbb{Z}^*_n$. And so, $f_{n,e} : \mathbb{Z}^*_n \to \mathbb{Z}^*_n$, so $f$ is a permutation function.

And in our RSA function $f$ and values $x$ and $c$, we assume that $n$, $e$, and the method for obtaining $f_{n,e}(x')$ for some value $x'$ is public; but the $x$ value that yields $c$, the primes $p$ and $q$ that make up $n$, and the value $d$ that would give $x$ from $c$ (since $c^d = (x^e)^d = x \mod n$) are all private information.

Notice how things differ here for $f$ in RSA compared to the $f$ used in the modular exponentiation section (in spite the fact that they both involve modulus exponentiation). With the modular exponentiation section, the exponent is secret and the base is public, whereas in RSA, the base is secret and the exponent is public.

And, assuming $n$ is a $k$-bit number, (and therefore, so is $x$ and $e$), from an arbitrary $x$, $f_{n,e}(x) = x^e \mod n$ can be computed in time polynomial in terms of $k$, for reasons mentioned in the Modular Exponentiation section. Therefore, $f_{n,e}$ is easy to compute.

We also notice that the best known way to invert RSA involves factoring $n$ into its primes (which allows one to learn $\varphi(n)$, which allows one to figure out a $d$, and therefore get $x$ from $c$, as mentioned earlier), and this is believed to be hard to do (no PPT algorithm known for it yet). In particular, other common methods for inverting $f_{n,e}$, like finding out $\varphi(n)$ or $d$ directly have been shown to be just as hard as factoring $n$. Therefore, $f$ is believed to be hard to invert.[4]

Additionally, since we just argued that inverting RSA is easy with either the factorization on $n$ (or the value $d$ above), RSA is a good TDP candidate.

**Conjecture 6** *RSA is a* TDP *(family).*

## 7 Things to Consider

After seeing all this info on OWF, OWP, and TDP. Here are three important principles in Cryptography to consider.

1. **Cryptography based on general theory**

   There are plenty of candidates for OWF, OWP, and TDP with believed hardness to invert based on differing reasons. Therefore, even if we figure out how to easily compute the prime factorization of any number (which for example will break RSA as a TDP candidate), we can still use something else as a candidate. Thus, there

[4] Even though breaking RSA is believed to be slightly easier than breaking factoring, RSA resisted many attacks and believed to be very hard to invert too.

are a lot of merits in basing cryptographic constructions on such general primitives like OWF's, OWP's or TDP's. Not only this gives us protection against breaking some specific function believed to be OWF, etc., but also allows us to distill which properties of a given function are crucial to make the construction work.

2. **Cryptography based on specific function assumptions**

   Sometimes, we can get more effective schemes if we assume certain function properties, like if $f(a) * f(b) = f(a + b)$ (which is true with the function $f$ used in the Modular Exponentiation section). Thus, for the purposes of efficiency and simplicity, schemes based on specific functions are also extremely useful in practice. Of course, these constructions are also less general than general constructions.

3. **Specific things give rise to actual implementations**

   Finally, even implementing general constructions using specific candidates gives rise to the actual real-life systems, showing ho our general theory can be applied in practice.

In the next section we return to OWFs, OWPs and TDPs, and give a sample *general* application for these concepts.

## 8 Application of OWF's: Password Authentication

Assume I wish to login to a server with my password $x$, but I don't want my server to store $x$, since the server's contents might be open to the world. (An in fact, in one of my previous jobs, the server not only stored passwords, but did so in a text file that was marked public to the world... oops!)

To solve this issue, I compute $f(x) = y$ where $f$ is OWF, and I tell the server only about $y$ and $f$. (So the server doesn't store $x$.) When I login, I give the server $z$, and the server checks if $f(z) = y$. Since $f$ is OWF, a hacker can't figure out a $z$ such that $f(z) = y$ easily (since $f$ being OWF implies that any algorithm $A$ where $A(y) \to z$ and $f(z) = y$ doesn't run in PPT), so this system is secure.

However, some problems do occur:

1. How can we be sure $x$ is uniform?

   Too many times, we pick passwords based on our login name, real name, birthday, family, friends, the new fancy word used by George W. Bush, etc. Or we pick passwords based on dictionary words, or our passwords are simply too short in length. (A hacker can easily write a program to guess all four-character password combinations until it gets a correct one.) Thus for many, $x$ is not truly randomly chosen and might be easily found.

   Turns out, there are tools (called commitment schemes, extractors and password-atuhenticated key exchange) to somewhat overcome this problem. Except for commitments, we will not have time to talk about them in this course though.

2. Although $x$ is not stored on the server, how can we trust that a hacker never sees $x$?

For example, if I connect to this server via `telnet`, Eve can run a program to identify myself, snoop onto my session, and read the text of everything I type during this session. From this, she can figure $x$ based on what I typed. One way to overcome this is to use `ssh` (secure shell), because `ssh` encrypts the text that I type before it is being sent over the Internet. Therefore, Eve's snooping will only get her an encryption of the text I typed, and she's forced to attempt and decrypt in order to get $x$. Thus `ssh` is much safer than telnet, and is the reason why various companies only allow people to connect remotely via `ssh` and not `telnet`. However, there might exist ways snoop the password by other ways, so even this also isn't a perfect solution.

It turns out there are advanced techniques (called identification schemes, zero-knowledge arguments, and proofs of knowledge) which will solve this problem as well. In the next section we will present a much simpler simple way to partially solve this problem.

However, assuming a hacker can only see the server's storage $y$, cannot see typed password $x$, and assuming that $x$ is truly randomly chosen, we immediately see that the password system mentioned earlier is secure assuming OWF's exist.

## 9 Application of OWP's: S/Key System

Taking the notion of password authentication a step further, let's suppose that a server keeps track of $T$ logins you make onto it, and changes the information it stores every time you login, and what you'd have to type to authenticate yourself will also keep changing accordingly. (This would prevent a hacker from getting being able to impersonate you after snooping one of your authentications.) Here is the way to do it. Take a *random* $x$, and compute for each $i$ from 0 to $T$, $y_i = f^{T-i}(x)$ (so $y_i = f(f \dots f(x) \dots)$) applied $(T - i)$ times) where $f$ is OWP. Notice, being a permutation ensures that we can apply $f$ to itself many times. We give the server only the last result $y_0$ (and the public function $f$), and nothing else. Note $y_0 = f^T(x)$.

Then, the first time we login, the server asks for $y_1$, and will check if $y_0 = f(y_1)$. If we correctly give it $y_1$ where $y_1 = f^{T-1}(x)$, then the server replaces $y_0$ with $y_1$.

The next time we login, the server will ask for $y_2$, and will check if $y_1 = f(y_2)$. Correctly giving $y_2$, where $y_2 = f^{T-2}(x)$, will make the server replace $y_1$ with $y_2$.

And so on, until the $T$-th login, where the server asks for $y_T$, and checks if $y_{T-1} = f(y_T)$. Correctly giving $y_T$, where $y_T = f^{T-T}(x) = f^0(x) = x$, ends the chain of $T$ logins, and we will have to start this process over (i.e., we give the server a new set of $y_0$ and $f$ and redo this).

Notice that in the process described above, we start with $y_0 = f^T(x)$. And, for each $i$, $y_i = f(y_{i+1})$, therefore, $y_{i+1} = f^{-1}(y_i)$, so we're inverting $f$ once at each step to get the next correct $y_j$ value (which the server stores for next time, and the server and hackers can't figure out any inverse until you type it in). And, we start over once we end up giving $x$ to the server.

Notice here that after each login, the server not only changes the value it stores, but stores the value you just entered, so the server (and the hacker) is always "a step behind". While the S/Key system intuitively seems secure, let us actually proof this *formally*! We

first need to learn learn about general $T$-time password authentication systems... Later we will show that S/Key is indeed such a system.

DEFINITION 9 [$T$-time Password Authentication System] A $T$-time Password Authentication System is a $T$-period protocol between the PPT server and the PPT user. First, given the "security parameter" $k$, the user and the server engage in the private setup protocol (which runs in time polynomial in $k$), after which the user has some secret information $SK$, and the server has some public information $PK$. From now on, all the server's storage (which is initially $PK$) is made public, and the user and the server now engage in $T$ login protocols. After each such protocol, the server decides whether to accept or reject the user. We require the following:

1. If the server talks to the real user (who knows $SK$ and is honest), the server must accept that user during any of the $T$ logins.

2. For any number of logins $t$, even if a hacker sees information the user uses to login to the server from his previous $(t-1)$ logins (in addition to the server's storage), the Probability that the hacker can impersonate the user's $t$-th login is at best $\mathsf{negl}(k)$, where $k$ is the security parameter (here roughly size of the data the hacker needs to guess).

   More formally, any $t < T$ and any PPT $A$, the probability that the server accepts $A$ at $t$-th login is $\mathsf{negl}(k)$. The latter probability is taken over the randomness used to setup the system, the randomness used by the user to login the first $(t-1)$ times, the possible randomness of the server to verify all the logins, and the randomness of $A$.

♢

It sounds like the S/Key system might be a good example of a $T$-time Password Authentication System. In fact it is, but only if the $f$ used is OWP. (Just having $f$ being OWF isn't good enough. See the homework...) Now, let's prove that if $f$ is OWP, S/Key is a $T$-time Password Authentication System.

**Theorem 7** *$\forall$ OWP $f$, S/Key is a $T$-time Password Authentication System.*

**Proof:** Assume there's some OWP $f$ such that S/Key is not secure. We'll show that this assumption leads to the fact that $f$ is not OWP, which is a contradiction.

Since S/Key is not secure for this $f$, there is some period $t$ and some PPT $A$ that achieve the following. Let $x \leftarrow \{0,1\}^k$ be chosen at random in the setup phase, let $y_i = f^{T-i}(x))$, so that the server stores $y_0$. Seeing first $(t-1)$ logings of the real user together with the server's storage gives $A$ exactly $y_0, \ldots, y_{t-1}$. A success for $A$ at time $t$ means that $A(y_{t-1}, \ldots, y_0) \rightarrow y'_t$ and $f(y'_t) = y_{t-1}$. Since $f$ is a permutation, success means that $y'_t = y_t$. To summarize, the assumption that S/key is insecure at time period $t$ implies:

$$\Pr[\ y'_t = y_t \mid x \leftarrow \{0,1\}^k,\ y_i = f^{T-i}(x),\ \forall\ 0 \le i \le T,\ y'_t \leftarrow A(y_{t-1}, \ldots, y_0)\ ] = \epsilon \qquad (1)$$

where $\epsilon$ is non-negligible.

With this in mind, we construct a new PPT adversary $\tilde{A}$ who will invert OWP $f$ with non-negligible probability (actually, the same $\epsilon$). $\tilde{A}$ will be given $\tilde{y} = f(\tilde{x})$, where $\tilde{x}$ was

chosen at random from $\{0,1\}^k$. The goal of $\tilde{A}(\tilde{y})$ is to come up with $\tilde{x}$. Naturally, $\tilde{A}$ will use the hypothetical $A$ to achive this (impossible) goal. Specifically,

$\tilde{A}$: On input $\tilde{y}$ run $z \leftarrow A(\tilde{y}, f(\tilde{y}), \ldots, f^{t-1}(\tilde{y}))$ and output $z$.

In other words, $\tilde{A}$ "fools" $A$ into thinking that $\tilde{y}$ was the password $y_{t-1}$ at period $(t-1)$, $f(\tilde{y})$ was the password $y_{t-2} = f(y_{t-1})$ at period $(t-2)$, and so on. Notice, since computation of $f$ is poly-time and $A$ is PPT, $\tilde{A}$ is PPT as well. Also, since $f$ is a *permutation* and both $x$ and $\tilde{x}$ were chosen at random, the distribution

$$D_0 = \langle f^{T-t+1}(x), \ldots, f^T(x) \rangle = \langle y_{t-1}, \ldots, y_0 \rangle$$

really expected by $A$ in (1), is the same as the distribution

$$D_1 = \langle f(\tilde{x}), \ldots, f^t(\tilde{x}) \rangle = \langle \tilde{y}, \ldots, f^{t-1}(\tilde{y}) \rangle$$

which $A$ received from $\tilde{A}$. Thus, our $\tilde{A}$ will succeed in finding the preimage of $\tilde{y}$ (which is necessarily $\tilde{x}$) with the same probability $\epsilon$ that $A$ find the preimage of $y_{t-1}$. But this violates the definition $f$ being hard to invert. So $f$ is not OWP, and we get a contradiction. □

## 10 Application of TDP's: Weak Public-Key Encryption

This is our original motivation to study TDP's. Namely, define the following public-key "encryption" scheme. The quotes are due to the facts that the encryption achieved will only satisfy a truly minimal (and insufficient) notion of security. Still, it is a good start.

The public key $PK$ will be the description of the TDP $f$ itself (i.e., the public key output by the TDP key generation algortihm Gen). The secret key $SK$ will be the corresponding trapdoor information output by Gen that makes $f$ easy to invert. To encrypt $m$, Bob sends Alice $c = f(m)$. Alice decrypts $c$ using the trapdoor $SK$.[5] The security of TDP's says that $f$ is hard to invert if $m$ is *random* in $\{0,1\}^k$ (more generally, whatever the domain of $f$ is). Thus, the only thing we can say about this encryption is that **Eve cannot completely decrypt encryptions of random messages**. This is a very weak notion, also called "one-wayness", but for encryption!

DEFINITION 10 An encryption scheme $(G, E, D)$ is *one-way secure*, if $\forall$ PPT Algorithm $A$,

$$\Pr(m' = m \mid (PK, SK) \leftarrow G(1^k);\ m \leftarrow \mathcal{M};\ c \leftarrow E_{PK}(m);\ m' \leftarrow A(c, PK, 1^k)) \leq \mathsf{negl}(k)$$

♢

Now, it is trivial to see that

**Lemma 1** *If $f$ comes from a* TDP *family, then the above encryption scheme is one-way.*

Unfortunately, one-wayness is only a very weak security notion for encryption. For example, one-wayness does not ruled out the following:

[5] More formally, we can define encryption scheme $(G, E, D)$ using TDP collection (Gen, Eval, Sample, Invert) by letting $G =$ Gen, $E_{PK}(m) =$ Eval$(m, PK)$ (here $m$ belongs to the domain of the TDP, which we assume is $\{0,1\}^k$ for simplicity), $D_{SK}(c) =$ Invert$(c, SK)$.

- Maybe Eve can get most of the message $m$ (but not all of it). Say, Eve might get half of the message. To see that this threat is actually possible, take any great TDP $f'$. Define $f(x_1, x_2) = (f'(x_1), x_2)$, where $|x_1| = |x_2| = k/2$. It is very easy to see that $f$ is a TDP such that the "encryption" of $m = (x_1, x_2)$ reveals half of the bits of $m$ (namely $x_2$). This is a contrived example, but even for "natural" $f$'s (like RSA) it turns out we can get some information about $m$ from $f(m)$. In fact, one such "information" is the value $f(m)$ itself!

- Nothing is said if $m$ is not random. For example, if an army base uses encryption to communicate with a mobile unit, and the only two messages the base will tell the unit is "attack" or "retreat," then the enemy unit can compute the values for $f(m)$ when $m$ is "attack" or "retreat," and based on these values, figure out $m$ from the ciphertext $c$.

Thus, this encryption leaves much to be desired, but is a non-trivial start.

## 11 Criticisms against OWF, OWP, TDP

Motivated by the above and the previous example, we can put forward the following criticism to the notions of OWF, OWP, and TDP:

1. When input $x$ is not random, how can we be sure the system is secure?

   See the "attack"/"retreat" example above for the demonstration. (I.e., if $x$ can only be a few values, we can compute the $f$-map for all these few values, and use this to learn $x$). It turns out that this issue can be solved. Essentially, we will *design* our application so that $x$ is always chosen at random! We will see how this is possible on later examples.

2. Viewed an as "encryption", the function $f$ could reveal a lot of partial information about $x$. See the pathological example of $f(x_1, x_2) = (f'(x_1), x_2)$ above. More realistically, take the Modular Exponentiation candidate example earlier, where $f_{p,g}(x) = g^x \mod p = y$. It turns out that from $y$, the last bit of $x$ can be efficiently extracted, despite the hardness to extract the entire $x$. (I.e., we can determine if $x$ is even or odd). A proof of this will be shown on the next lecture.

## 12 Ways out of the Criticism against OWF, OWP, and TDP

But alas, there are few twists we can try in order to avoid the criticisms mentioned above...

1. One way is to design an encryption function $f(x)$ hide all info about $x$. Unfortunately, this is exactly our goal of designing secure encryption! Thus, we came back to where we started. A new idea is needed to break out! Notice, however, that it is clear that no *deterministic* $f$ can achieve this goal (see the "attack"/"retreat" example again; more trivially, $f(x)$ is "information" about $x$). Thus, we know that such $f$ must be *probabilistic*!

2. If $x$ is only a few values, we can create a function $g(x) = f(x,t) = z$ where $t$ is a counter increased by 1 each time a new message is sent. The German ENIGMA machine during World War II used a technique like this. Fortunately, this had bad consequences for the Germans.[6] Unfortunately, while useful in practice, this technique lacks formal justification, at least in this simple way. Indeed, if $x$ is not random, $(x,z)$ is not random as well, so we still cannot use our definitions. More importantly, $f(x,z)$ might still allow one to recover most (if not all) of the bits of $x$.

3. How about requiring $f(x)$ to *"completely hide"* information about some function $h(x)$. In other words, standard definition tells us that $f(x)$ does not allow the hacker to get $x$ completely, but may allow to get a lot of partial information about $x$. So maybe we can pin-point some partial information $h(x)$ (i.e., whether $x$ is even or odd) which still remains completely hidden from the adversary who knows $f(x)$. In other words, $f(x)$ does not allow the adversary to learn *anything* about $h(x)$. Put yet differently, **use $f(x)$ to "encrypt" $h(x)$!!!** We will see, this is exactly out golden way out...

[6] This shows that sometimes cryptographic ignorance could be of use to the humanity. Hopefully, this argument is outdated by now: ignorance should never be good!

| CSCI-GA.3210-001<br>MATH-GA.2170-001 | **Introduction to Cryptography** | Feb 8, 2012 |
|---|---|---|
| | Lecture 3 | |
| *Lecturer: Yevgeniy Dodis* | | *Spring 2012* |

This lecture mainly discusses some basic facts about $\mathbb{Z}$, $\mathbb{Z}_p$, $\mathbb{Z}_n$. Special interest is given to the computational complexity of various operations defined on these sets. In $\mathbb{Z}$, we explain the extended Euclid algorithm in detail, which finds its application along the way. In $\mathbb{Z}_p$, we study the existence and computational complexity of finding inverses, exponential, discrete log, solving various polynomial equations and extracting square roots. In $\mathbb{Z}_n$, Chinese Remainder theorem is discussed, which enables to reduce many problems from $\mathbb{Z}_n$ to $\mathbb{Z}_p$. Several candidate OWFs are presented along the way, including factoring, discrete log, RSA and modular squaring.

## 1 Facts in $\mathbb{Z}$

Recall, $\mathbb{Z} = \{0, \pm 1, \pm 2, \ldots\}$, $\mathbb{Z}_+ = \{1, 2, \ldots\}$, and $a|b$ means that $a$ divides $b$. The following Lemma is well known.

**Lemma 1** *Addition, multiplication in $\mathbb{Z}$ can be done in polynomial time (with respect to length of operands). Moreover, for any $a > b > 0$ one can find in polynomial time unique $\tau$ and $0 \le q < b$ such that $a = \tau b + q$. On the other hand, exponentiation $\exp(a, b) = a^b$ cannot be perfoprmed in polynomial time.*

The latter part follows since if $a = b = 111...1$ ($k$-bit long), then $a^b$ is roughly $\frac{1}{2}k \cdot 2^k$ bit long, so it is even impossible to write the answer down!

Definition 1 [GCD] For $a, b \in \mathbb{Z}_+$, define $gcd(a, b) := \max\{d \ge 1 \ : \ d|a \text{ and } d|b\}$. $\diamondsuit$

**Proposition 2** $gcd(a, b) = \inf\{a\tilde{u} + b\tilde{v} > 0 \ : \ \tilde{u}, \tilde{v} \in \mathbb{Z}\}$.

It will be proven as a special case of the Theorem below.

**Theorem 1 (Extended Euclid Algorithm)** *Given $a$, $b$, one can compute in polynomial time the value $gcb(a, b)$. Moreover, in polynomial time one can find integers $u$ and $v$ such that $gcd(a, b) = au + bv$.*

**Proof:** Without loss, we assume that $a > b > 0$. Assume that $b$ is $k$-bit long and let $b = q_0$. We carry out division in $\mathbb{Z}$.

$$a = b\tau_1 + q_1, \ 0 \le q_1 < b \tag{1}$$

It is easy to verify that $gcd(a, b) = gcd(b, q_1)$. Now iterate:

$$b = q_1\tau_2 + q_2, \ 0 \le q_2 < q_1 \tag{2}$$

and so on. We stop when we get $q_{s+1} = 0$. Then

$$gcd(a, b) = gcd(b, q_1) = gcd(q_1, q_2) = \ldots = gcd(q_{s-1}, q_s) = q_s \tag{3}$$

We only need show that $s$ is bounded by polynomial of $k$.

We claim that for any $i$, $q_i \geq 2q_{i+2}$.[1] Indeed, either we right away have $q_i \geq 2q_{i+1} \geq 2q_{i+2}$, or otherwise $q_i \geq q_{i+1} + q_{i+2} \geq \frac{q_i}{2} + q_{i+2}$, so that anyway $q_i \geq 2q_{i+2}$. Thus every two iterations $q_i$ decreases by a factor greater than 1, i.e. we are done after a linear number of iterations.

□

EXAMPLE. Say, $a = 14$, $b = 10$. Then $14 = 1 \cdot 10 + 4$, $10 = 2 \cdot 4 + 2$, $4 = 2 \cdot 2 + 0$. Thus, $gcd(14, 10) = 2$. Moreover, going back from the next-to-last equation to the first equation, we get

$$2 = 10 - 2 \cdot 4 = 10 - 2 \cdot (14 - 1 \cdot 10) = (-2) \cdot 14 + 3 \cdot 10$$

so $u = -2$, $v = 3$.

Now we briefly discuss the primes.

**Theorem 2 (Prime Number Theorem)** *When $k$ is large enough, there are roughly $\frac{2^k}{k}$ primes in $[1, 2^k]$ (up to some small constant factor).*

From this theorem, we know that for some constant $c$,

$$\Pr[\text{random } k\text{-bit integer is prime}] \approx \frac{c}{k}$$

Thus, after linear number of trials we expect to sample a prime number. Can we test if a given number $n$ is prime? It turns out the answer is yes. There are many efficient *probabilistic* primality tests (such as Miller-Rabin or Solovay-Strassen), and, recently, even a (slower) deterministic primality test (called AKS after Agrawal, Kayal and Saxena) was found. Thus,

**Theorem 3** *One can test in polynomial time if a given $k$-bit number $n$ is prime. Hence, one can sample a* random *$k$-bit prime in expected polynomial time.*

On the other hand, given two *random* $k$-bit primes $p$ and $q$, we do not know how to compute $p$ and $q$ from their product $n = pq$ in probabilistic polynomial time (in $k$). This is the famous *factoring problem*. It is widely believed that

**Conjecture 4** *Integer multiplication of two random $k$-bit primes is a* OWF.

[1] A slightly trickier analysis shows in fact that $q_i \geq \frac{\sqrt{5}+1}{2} \cdot q_{i+1}$.

## 2 Facts in $\mathbb{Z}_p$

$\mathbb{Z}_p = \{0, 1, 2, ...p-1\}$. We can define addition and multiplication modulo $p$. It is well known that both of these operations are commutative and associative. Moroeover, addition forms a *group* with identity 0 and inverse $(-a)$ defined as $p - a$. We can also define the *multiplicative subgroup* $\mathbb{Z}_p^* = \{a \in \mathbb{Z}_p \mid gcd(a, p) = 1\}$. It is easy to see that $\mathbb{Z}_p^*$ is closed under multiplication, has identity 1, and the existence of the inverse is easily derived from the Extended GCG algortihm (Theorem 1). Indeed, take any $a \in \mathbb{Z}_p^*$. Since $gcd(a, p) = 1$, there exists $u$, $v$, such that $au + pv = 1$. Then $a \cdot u \equiv 1 \bmod p$. So $(u \bmod p)$ is the needed inverse of $a$.

Example. For example, $gcd(5, 11) = 1$ and $1 = 11 - 2 \cdot 5$. Thus, $5^{-1} \bmod 11 \equiv -2 \equiv 9$. Indeed, $9 \cdot 5 \bmod 11 \equiv 45 \bmod 11 = 1$.

For the remainder of this section, we will be interested in a special case when $p$ is *prime*. In this case, $\mathbb{Z}_p^* = \{a \in \mathbb{Z}_p \mid gcd(a, p) = 1\} = \{1, \ldots, p-1\}$ has all $(p-1)$ of $\mathbb{Z}_p$'s non-zero elements, and every element $a$ in $\mathbb{Z}_p^*$ has a unique multiplicative inverse $u \equiv a^{-1} \bmod p$ (i.e. $u$ satisfying $au \equiv 1 \bmod p$). Mathematically speaking, this means that

**Fact 5** *$\mathbb{Z}_p$ is a* field *if $p$ is prime.*

Next, we mention the famous Fermat's little theorem.

**Theorem 6 (Fermat)** *For any $a \in \mathbb{Z}_p^*$, $a^{p-1} \equiv 1 \bmod p$, and thus, for integers $a, b$,*

$$a^b \bmod p \equiv (a \bmod p)^{b \bmod (p-1)} (\mathrm{mod} p)$$

**Proof:** The result follows from the Lagrange's theorem that the order of every element in a finite group (in this case $\mathbb{Z}_p^*$) divides the size of the group (in this case $p-1$). Recall, the *order* of $a$ is the smallest integer $i > 0$ s.t. $a^i \equiv 1$. Since any such $i$ divides $(p-1)$, we get $a^{p-1} \equiv (a^i)^{(p-1)/i} \equiv 1 \bmod p$. □

Example. $58^{42} \bmod 11 \equiv (58 \bmod 11)^{42 \bmod 10} \equiv 3^2 \bmod 11 = 9$.

Getting to the computational side, it is easy to see that addition, multiplication can be done in polynomial time (with respect to bits of p). In contrast to $\mathbb{Z}$, we have

**Theorem 7** *$exp : a, b \mapsto a^b \bmod p$ can be done in polynomial time, with respect to bits lengths of $a, b, p$.*

**Proof:** Denote $k$ is the bits length of $p$, and by Fermat's theorem assume that $a, b \leq p$ are $k$-bit as well (if needed, replace $a$ by $a \bmod p$ and $b$ by $b \bmod (p-1)$). Now we can use the **repeated squaring technique**,

$$\begin{aligned} a^{\sum_{i=0}^{k-1} b_i 2^i} &= (a)^{b_0} \cdot (a^2)^{\sum_{i=1}^{k-1} b_i 2^{i-1}} \\ &= (a)^{b_0} (a^2)^{b_1} (a^4)^{\sum_{i=2}^{k-1} b_i 2^{i-2}} = \ldots \\ &= (a)^{b_0} (a^2)^{b_1} (a^4)^{b_2} \ldots (a^{2^{k-1}})^{b_{k-1}} \end{aligned}$$

Now it is easy to see the computing time is bounded by $O(k^3)$, since it takes $O(k^2)$ time to square in $\mathbb{Z}_p$, and we need to do $k$ repeated squarings to successively compute $(a^{2^i} \bmod p)$ for all $1 \leq i < k$. □

Next, we already observed that finding inverses can be done in polynomial time as well. As a simple generalization,

**Proposition 3** *Linear system in $\mathbb{Z}_p$ can be solved in polynomial time.*

More importantly, it is known that

**Theorem 8** *$\mathbb{Z}_p^*$ is a* cyclic group*, which means that $\exists g \in \mathbb{Z}_p^*$, such that $\mathbb{Z}_p^* = \{1, g, g^2, ..., g^{p-2}\} = \{1 \dots p-1\}$. Any $g$ as above is called a* generator *of $\mathbb{Z}_p^*$.*

In other words, $(p-1)$ successive powers of such $g$ "generate" — without repetition and exactly once — all the elements of $\mathbb{Z}_p^*$. From this theorem, we see that Fermat's theorem cannot be improved by replacing $p-1$ by a smaller number.

EXAMPLE. 3 is a generator of $\mathbb{Z}_7$, while 2 is not. Indeed, $\{3^0, 3^1, 3^2, 3^3, 3^4, 3^5\} = \{1, 3, 2, 6, 4, 5\} = \{1, 2, 3, 4, 5, 6\}$, while $2^3 \equiv 1 \bmod 7$.

How many generators does $\mathbb{Z}_p^*$ have? Is it easy to find some? Is a random element likely to be a generator? Let us begin to answer such questions. Jumping ahead, for any (not necessarily prime) integer $n$ define $\varphi(n) = |\{a \in \mathbb{Z}_n \mid gcd(a, n) = 1\}|$. Notice, for the prime $p$, $\varphi(p) = p - 1 = |\mathbb{Z}_p^*|$. Also, it is relatively easy to show by counting that for any $n$, $\varphi(n) \geq n/\log\log n$. Now fix any generator $g$ in $\mathbb{Z}_p^*$ and take any $y \in \mathbb{Z}_p^*$. Let us write $y = g^x$, for some $x$ (we can do it since $g$ is a generator). Then

$$\begin{aligned} y \text{ is } \textit{not} \text{ a generator} &\iff y^b = 1 \text{ for some } 0 < b < p-1 \\ &\iff g^{bx} = 1 \text{ for some } 0 < b < p-1 \\ &\iff bx \equiv 0 \bmod (p-1) \text{ for some } 0 < b < p-1 \\ &\iff gcd(x, p-1) > 1 \end{aligned}$$

Thus, we get a characterization of when $y$ is a generator in terms of the *discrete log* of $y$ base $g$. In particular,

**Theorem 9** *The number of generators of $\mathbb{Z}_p^*$ is $\varphi(p-1) \geq \frac{p}{\log\log p}$.*

The above proof suggests a plausible way to pick a generator: simply sample it at random from $\mathbb{Z}_p^*$. Unfortunately, we do not know of an efficient way to test if a given $y$ is a generator, given only $p$ alone. On other other hand, we claim that it is easy to test if $y$ is a generator given a factorization of $p-1$. Indeed, assume $(p-1) = \prod_{i=i}^{t} q_i^{\alpha_i}$, where $q_i$ are all prime. Notice, as all $q_i \geq 2$, $2^{k+1} > p \geq 2^t$, so $t \leq k$. From Lagrange theorem, the order of $y$ must divide $(p-1)$, and is strictly less than $(p-1)$ if and only if $y$ is *not* a generator. But then the order must divide at least one of "most immediate divisors" $(p-1)/q_1, \dots, (p-1)/q_t$. Thus,

$$y \text{ is } \textit{not} \text{ a generator} \iff \exists\ 1 \leq i \leq t \text{ such that } y^{(p-1)/q_i} \equiv 1 \bmod p$$

EXAMPLE. If $p = 7$, $p - 1 = 6 = 2 \cdot 3$, so $y$ is a generator if and only if $y^2 \bmod 7 \neq 1$ and $y^3 \bmod 7 \neq 1$. Then, 3 is a generator since $3^2$ and $3^3$ are different from 1 modulo 7, while 2 is not, since $2^3 \bmod 7 = 1$.

And since $t \leq k$ and exponentiation can be done in polynomial time as well, we can test if a given $y$ is a generator in polynomial time *given the factorization of* $(p-1)$. Luckily, it turns out that we can sample a random $k$-bit number $v$ together with its factirization (famous result due to Bach and simplified by Kalai). By picking such $v = p-1$ at random, testing if $p = v+1$ is a prime, then picking a random $g \in \mathbb{Z}_p$ and using the factorization of $v = (p-1)$ to test if $g$ is the generator, we get

**Theorem 10 (Bach,Kalai)** $\forall\ k \geq 1$ *there is a PPT sampling algorithm that returns:*

- *A random $k$-bit prime $p$.*
- *A random generator $g$ of $\mathbb{Z}_p^*$.*
- *Complete factorization of $(p-1)$ (this can be useful sometimes).*

We now study the quadratic equations $x^2 \equiv a \bmod p$ in $\mathbb{Z}_p$.

**Proposition 4** $\forall a \neq 0$, *the equation below either zero or exactly two solutions $x$. In the latter case the two solutions are of the form $\pm w$ for some $w \in \mathbb{Z}_p^*$.*

$$x^2 = a \bmod p \tag{4}$$

**Proof:** If $x^2 = a$, then $(-x)^2 = a$ as well. Notice, since $p$ is an odd prime and $x \neq 0$, $x \not\equiv -x \bmod p$ (or precisely: $x \neq p - x$ over integers). Thus 1 root is impossible. Also, if $x^2 = y^2 \bmod p$, then $(x-y)(x+y) = 0$, so $x = \pm y$, so more than 2 roots are impossible. □

DEFINITION 2 $a \in \mathbb{Z}_p^*$ is called **QR** (Quadratic Residue) iff $x^2 = a$ has two solutions. ♢

We seek characteristic of QR.

**Lemma 5** *Suppose $g$ is generator of $\mathbb{Z}_p^*$, $a \in \mathbb{Z}_p^*$ and $a = g^z$. Then*

$$a \text{ is QR} \iff z \text{ is even} \iff a^{\frac{p-1}{2}} \equiv 1 \bmod p$$

**Proof:** If $z = 2w$ is even then $a = g^z = (g^w)^2$ is QR. Conversely, if $a = (g^w)^2$ is QR, then $z = 2w \bmod (p-1)$. Since $(p-1)$ and $2w$ are even, so is $2w \bmod (p-1)$.

For the second part, if $z = 2w$ is even then $a^{(p-1)/2} = g^{(p-1)w} = 1 \bmod p$, by Fermat's theorem. Conversely, if $(g^z)^{(p-1)/2} = 1$ and since $g$ is a generator, then $z \cdot (\frac{p-1}{2}) \equiv 0 \bmod (p-1)$, which means that $z \cdot (\frac{p-1}{2}) = w(p-1)$ (over integers for some integer $w$), so that $z = 2w$ is even. □

As corollary, we find that exactly half of $\mathbb{Z}_p^*$ elements are QR. The last result also inspires us to introduce

DEFINITION 3 [Legendre's symbol] Suppose $p$ is prime, and $a \in \mathbb{Z}_p^*$. Define Legendre's symbol $\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}} \bmod p$. ♢

Notice, by Fermat's theorem, $\left(\frac{a}{p}\right)^2 \equiv 1 \bmod p$, i.e. $\left(\frac{a}{p}\right) \in \{1, -1\}$. So we can rephrase the above result as: $a$ **is QR iff** $a$**'s Legendre symbol is** $+1$.

Note that the definition itself gives us an efficient way to compute Legendre symbol (apply the repeated squaring technique). As the result, it is easy to verify if $a$ is QR in $\mathbb{Z}_p^*$. What's more,

**Theorem 11** *if $p$ is prime, then $x^2 \equiv a \bmod p$ can be solved in PPT.*

Note that we have a very simple explicit solution $x = \pm a^{\frac{p+1}{4}} \bmod p$ if $p \equiv 3 \bmod 4$ (why?). When $p \equiv 1 \bmod 4$, things are slightly more complicated, but again can be done in polynomial time. In fact, algebraic equation of degree $d$ in $\mathbb{Z}_p$ can be solved in PPT (in $d$). We don't give proof here.

EXAMPLE. Say, we try to solve $x^2 = 2 \bmod 7$. First, 2 is QR iff $\left(\frac{2}{7}\right) = 1$, which is true since $2^{(6-1)/2} \bmod 7 = 1$. Next, as $7 \equiv 3 \bmod 4$, the solutions are $\pm 2^{(7+1)/4} = \pm 2^2 \bmod 4 = \{3, 4\}$. Indeed, $3^2 \bmod 7 \equiv 4^2 \bmod 7 = 2$.

We would also like to mention a simple special case which can be solved efficiently: the equation $x^e = 1 \bmod p$, where $gcd(e, p-1) = 1$. In this case, we can find $d$, $v$ such that $ed + (p-1)v = 1$ by extended Euclid algorithm. It is easy to verify that $x^e \equiv a \bmod p$ has a unique solution $x = a^d \bmod p$ (why?). Thus, "RSA function" is easy over the primes. (Unfortunately, the above technique does not work for quadratic equations, since $(p-1)$ is even, and so $gcd(2, p-1) = 2$.)

Up to now, all the computation can be efficiently done in $\mathbb{Z}_p$. It is not always true. For example, computation of the **discrete logarithm** as below.

DEFINITION 4 [Discrete Logarithm] For $x$, $a$, we let the discrete logarithm (DL for short) of $x$ to be any $y$ satisfying $a^y \equiv x \bmod p$. ♢

It is a widely held belief that DL is computationally hard, especially when the "base" $a$ is a generator $g$ of $\mathbb{Z}_p^*$. Notice, in the latter case there a unique solution $x \in \mathbb{Z}_{p-1}$. More formally,

**Conjecture 12** *$g^x \bmod p$ is believed to be a* one-way permutation *from $\mathbb{Z}_{p-1}$ to $\mathbb{Z}_p^*$,[2] when $p$ is a random $k$-bit prime, and $g$ is a random generator of $\mathbb{Z}_p^*$.*[3]

**Remark 1** *Why should we choose $k$-bit prime $p$ at random? Well, although discrete log is believed to be hard for most primes $p$, it is certainly not hard for* all *$p$. For example, a very conveneint choise of prime if of the form $p = 2^k + 1$ (and it is believed that this is prime for many $k$'s), since then the order of $\mathbb{Z}_p^*$ is $p - 1 = 2^k$, which is a power of 2 (which leads to very efficient implementations). However, this efficiency also leads to a simple algorithm (which one?) that can compute discrete log for such primes. So one has to be careful when choosing $p$ for which discrete log is hard.*

Although computing $x$ given $y = g^x \bmod p$ is believed to be hard, $y$ reveals some partial information about $x$. For example, from Lemma 5 we see that $x$ is even iff $y^{(p-1)/2} \equiv 1 \bmod p$. Since $x$ is even iff $LSB(x) = 0$ (where $LSB$ denotes least significant bit), we get

**Lemma 6** *Given $y \in \mathbb{Z}_p^*$ (and $p$, $g$), there exists an efficient algorithm to compute $LSB(\log_g(y))$.*

[2] Indeed, although algebraically $\mathbb{Z}_{p-1}$ and $\mathbb{Z}_p^*$ appear distinct, they both identify with $\{1 \dots (p-1)\}$.

[3] Moreover, it stays one-way even given the factorization of $(p-1)$.

## 3 FACTS IN $\mathbb{Z}_n(n = pq,\ p,\ q$ ARE PRIME$)$

First, we introduce an important theorem enabling us to reduce problems from $\mathbb{Z}_n$, where $n$ is composite, to $\mathbb{Z}_p$, where $p$ is prime.

**Theorem 13 (Chinese Remainder Theorem)** *Let $m_1$,...,$m_k$ be pairwise relative prime. Denote $m = m_1 m_2 \cdots m_k$, then $\forall a_1 \in \mathbb{Z}_{m_1}$, ..., $\forall a_k \in \mathbb{Z}_{m_k}$, there exists unique $a \in \mathbb{Z}_m$, which can be computed in polynomial time, such that*

$$a \equiv a_i \ mod\ m_i,\ \forall i = 1,...,k. \tag{5}$$

**Proof:** Assume $a = \Sigma_i a_i u_i$, where $u_i$'s are left to decide. Take mod $m_k$, we see that it is sufficient

$$\begin{aligned} u_i &\equiv 1 \bmod m_i \\ u_i &\equiv 0 \bmod m_j\ ,\ \forall j \neq i \end{aligned}$$

Let us write $u_i = (\Pi_{j \neq i} m_j) v_i$ for unknown $v_i$. Then $v_i \equiv (\Pi_{j \neq i} m_j)^{-1} \bmod m_i$, whose existence is ensured by the pairwise relatively primality. It is easy to see that the $u_i$'s above satisfy the needed condition. □

By the theorem, we have one-to-one correspondence $\mathbb{Z}_m \to \mathbb{Z}_{m_1} \times ... \times \mathbb{Z}_{m_k}$, $a \mapsto (a_1, ..., a_k)$. We will often use this corresponding without further explanation, i.e. saying $(a_1, \ldots, a_n) \in \mathbb{Z}_m$ means the unique $a \in \mathbb{Z}_m$ satisfying $a \bmod m_i \equiv a_i$. The more important thing is, the addition and multiplication are preserved under this correspondence.

$$\begin{aligned} a + b &= (a_1 + b_1, ..., a_k + b_k) \\ a \cdot b &= (a_1 \cdot b_1, ..., a_k \cdot b_k) \end{aligned}$$

EXAMPLE. Say $m_1 = 7$, $m_2 = 11$, $m = 77$, and $a = 54$. Then computing $a_1$ and $a_2$ is trivial: $a_1 = a \bmod 7 \equiv 5$ and $a_2 = a \bmod 11 = 10$. The interesting direction is the converse. Gievn $(5, 10)$, how do we derive 54 back? First, using extended Euclid we get $1 = 2 \cdot 11 - 3 \cdot 7 = 22 - 21$. Thus, $22 = (1, 0)$ and $-21 = 77 - 21 = 56 = (0, 1)$. Hence,

$$(5, 10) \equiv 5 \cdot (1, 0) + 10 \cdot (0, 1) \equiv 5 \cdot 22 + 10 \cdot (-21) \equiv 110 - 210 \equiv -100 \bmod 77 = 54$$

If we let $\mathbb{Z}_m^* = \{a \in \mathbb{Z}_m \mid gcd(a, m) = 1\}$, we get that $\mathbb{Z}_m^* = \mathbb{Z}_{m_1}^* \times ... \times \mathbb{Z}_{m_k}^*$ under this correspondence as well. Indeed, the number is relatively prime to $m \iff$ its residues modulo all the $m_i$'s are relatively prime to the corresponding $m_i$'s. Notice, $\mathbb{Z}_m^*$ is a multiplicative subgroup of $\mathbb{Z}_m$, whose size was defined to be $\varphi(m)$ earlier. From the correspondence above, $\varphi(m) = \Pi_{i=1}^k \varphi(m_i)$. In particular, since the order of every element in $\mathbb{Z}_n$ divides the order of the group $\varphi(n)$, we get the following generalization of the Fermat's little theorem.

**Theorem 14 (Euler's Theorem)** *For any $a \in \mathbb{Z}_n^*$, $a^{\varphi(n)} \bmod n \equiv 1$. In particular, for $n = pq$, $a^{(p-1)(q-1)} \bmod n \equiv 1$.*

EXAMPLE. Say $n = 35$, so $\varphi(n) = 24$. Then $72^{50} \bmod 35 \equiv (72 \bmod 35)^{50 \bmod 24} \equiv 2^2 = 4$.

From now on, we will always talk about $\mathbb{Z}_m^*$ rather than $\mathbb{Z}_m$, since if we ever encounter an element in $\mathbb{Z}_m \backslash \mathbb{Z}_m^*$, we can factor $m$, which is believed to be hard. So, except for mathematically, they will never come up in the cryptographic applications we consider.

Also notice that via the Chinese Remainder Theorem and the discussion in earlier section, we know

**Theorem 15** *Addition, multiplication, exponentiation and taking inverses can be done in polynomial time in $\mathbb{Z}_m^*$.*

From now on, let us concentrate on numbers of the form $n = pq$, where $p, q$ — $k$-bit primes. Notice, $\varphi(n) = |\mathbb{Z}_n^*| = (p-1)(q-1)$ in this case. However, determining $\varphi(n)$ from $n$ is easily seen to be equivalent to factoring $n$ (simply solve a system $pq = n$, $(p-1)(q-1) = \varphi(n)$)! Thus, we see the first difference with the primes: the order of $\mathbb{Z}_n^*$ is provably hard to determine under the hardness of factoring assumption.

Let us now turn to solving various equations over $\mathbb{Z}_n^*$. Since QRs in $\mathbb{Z}_p$ have two roots, we know QRs in $\mathbb{Z}_n$ have two times two = four roots. As in $\mathbb{Z}_p$, we introduce some symbol to help us characterize characterize QR in $\mathbb{Z}_n$.

DEFINITION 5 [Jacobi's symbol] If $n = pq$, $p$, $q$ are two primes, then we define the Jacobi's symbol of $a = (a_1, a_2) \in \mathbb{Z}_n^*$ to be $\left(\frac{a}{n}\right) = \left(\frac{a_1}{p}\right)\left(\frac{a_2}{q}\right)$. The right hand items are Legendre symbols. ♢

It turns out that there exists a polynomial time algorithm that can compute $\left(\frac{a}{n}\right)$ even *without the factorization of $n$* into $p$ and $q$. We will not cover it though. How does it relate to QR?

**Proposition 7** *If $a$ is QR in $\mathbb{Z}_n$, then the Jacobi's symbol of $a$ is $+1$. However, the reverse does* **not** *necessarily hold.*

**Proof:** $a$ is QR in $\mathbb{Z}_n$, then so is true for $a_1 \in \mathbb{Z}_p$, $a_2 \in \mathbb{Z}_q$, and thus $\left(\frac{a_1}{p}\right) = \left(\frac{a_2}{q}\right) = +1$, proving the first part. For the second part, notice that the number of QR's is a quarter of $\mathbb{Z}_n^*$, while the number of elements with Jacobi symbol $+1$ is a half of $\mathbb{Z}_n^*$. Indeed, $(a_1, a_2)$ with $\left(\frac{a_1}{p}\right) = \left(\frac{a_2}{q}\right) = -1$ yield $a = (a_1, a_2)$ with $\left(\frac{a}{n}\right) = (-1)(-1) = +1$, even though this $a$ is not a quadratic residue. □

Now, given $a \in \mathbb{Z}_n^*$ with Jacobi symbol $+1$, can we actually determine if $a$ is a QR over $\mathbb{Z}_n^*$? (if $\left(\frac{a}{n}\right) = -1$, we know the answer is "no".) The answer is not only believed to be negative, but

**Assumption 16 (QR assumption)** *For a randomly chosen $n = pq$, a randomly chosen $a \in \mathcal{J}_n = \{b\text{:}\ \left(\frac{b}{n}\right) = +1\}$, and for arbitrary PPT algorithm $\mathcal{A}$,*

$$\Pr[\alpha = QR(a) \mid n = pq,\ a \leftarrow \mathcal{J}_n,\ \mathcal{A}(a, n) \to \alpha \in \{\pm 1\}] \leq \frac{1}{2} + \mathsf{negl}(k)$$

Thus, we can not do anything better than tossing a coin! As immediate corollary, we can expect that it is computationally hard in $\mathbb{Z}_n^*$ to solve equations $x^2 \equiv a \bmod n$ *without the factorization of* $n$.[4]

Hence we obtain a candidate for OWF, namely, SQ:

DEFINITION 6 SQ is the function defined in $\mathbb{Z}_n$ as: $x \mapsto x^2 \bmod n$. ♢

As an evidence that our assumption is reasonable, we cite a result due to Rabin:

**Theorem 17** *SQ is* OWF $\iff$ *FACTORING is hard.*

**Proof:** Assume SQ is easy to reverse with non-negligible probability $\varepsilon$, we construct an efficient algorithm to factorize $n$ with non-negligible probability $\varepsilon/2$.

- Pick up a random $a \in \mathbb{Z}_n$.
- Compute $b$ as $SQ(a^2 \bmod n)$. If this fails, exit.
- IF $(a \equiv \pm b \bmod n)$ THEN fail and exit;
- ELSE output $(gcd(a+b, n), \frac{n}{gcd(a+b,n)})$ as the needed factorization.

Notice, if $\varepsilon$ is the probability $SQ$ algorithm works, we claim we have probability a $1/2$ of factoring $n$ conditioned on $SQ$ succeeding, i.e. $\varepsilon/2$ probability of sucess overall (contradicting that factoring is hard). For the latter claim, it is easy to see that if $a = (a_1, a_2)$ (for unknown $a_1 \in \mathbb{Z}_p^*$ and $a_2 \in \mathbb{Z}_q^*$), the 4 square roots of $a^2$ are $a = (a_1, a_2)$, $-a = (-a_1, -a_2)$, but also some $c = (-a_1, a_2)$ and $-c = (a_1, -a_2)$. With probability $1/2$ the successful $SQ$ inverter will return $c$ or $-c$ (rather than $a$ or $-a$). In the first case, $gcd(a + c, n) = p$, and in the second $gcd(a - c, n) = q$. In both cases, we factor $n$. □

We also mention that Rabin's function is 4-to-1, since every square has four square roots. It turns out, we can get a candidate one-way *permutation* for a special $p$ and $q$. Namely, assume $p \equiv 3 \bmod 4$ and $p \equiv 3 \bmod 4$ (such $n = pq$ is called a *Blum's integer*), and let $Q_n$ denote the subgroup of quadratic residues modulo $n = pq$. In this case, it turns out (we omit the proof) that out of 4 square roots of any $a \in Q_n$ *precisely one* also belongs to $Q_n$. Put differently, the squaring function $SQ$ is *a permutation over* $Q_n$. Thus,

**Lemma 8** *If factoring Blum's integers is hard, then SQ is a* OWP *over quadratic residues modulo* $n$.

Finally, let us briefly talk about the RSA function $x^e \bmod n$, where $e \in \mathbb{Z}_{\varphi(n)}^*$. This function is clearly easy to invert if factoring is easy. Simply solve $x^e = y$ in $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$ (as explained in the previous section) and use the Chinese Remainder theorem. Alternatively, compute $d = e^{-1} \bmod \varphi(n)$ (which is easy with factorization, since $\varphi(n) = (p-1)(q-1)$), and notice that

$$y^d \bmod n \equiv x^{ed} \bmod n \equiv x^{ed \bmod \varphi(n)} \bmod n \equiv x^1 \bmod n = x$$

[4] Of course, given the factorization of $n = pq$, solving $x^2 \equiv a \bmod n$ is easy. Using CRT, write $a = (a_1, a_2)$, then find solutions $\pm b$ to $x^2 \equiv a_1 \bmod p$, $\pm c$ to $x^2 \bmod a_2 \bmod q$, and then use CRT backward to get the four solutions $(b, c), (-b, c), (b, -c), (-b, -c)$ to $x^2 \equiv a \bmod n$.

EXAMPLE. Say $n = 5 \cdot 11 = 55$, and $e = 7$ (which is relatively prime to $\varphi(55) = 40$). Turns out, $d = 7^{-1} \bmod 40 = 23$ (as $7 \cdot 23 = 161 \equiv 1 \bmod 40$). Now, $2^7 \bmod 55 = 128 \bmod 55 = 18$, and one can check that $18^{23} \bmod 55 = 2$. The way to do it is to compute $18^{23} \bmod 5 = 3^3 \bmod 5 = 2$, $18^{23} \bmod 11 = 7^3 \bmod 11 = 343 \bmod 11 = 2$, and then use CRT to derive $(2, 2) = 2$ indeed.

The famous RSA assumption states that

**Conjecture 18 (RSA Assumption)** *For randomly generated $n = pq$ and $e \in \mathbb{Z}^*_{\varphi(n)}$, $RSA_{n,e}$ is a* TDP *family.*

How does RSA assumption compare to the factoring assumption? We just pointed out that is factoring is easy, then so is RSA. The converse is not known to be true. Namely, for all we know RSA might be slightly easier than factoring. However, one can show that the *specific* way to break RSA by computing $d$ from $e$ and $n$ is equivalent to the hardness of factoring. However, maybe there are some *other*, perhaps less "natural" but more effective, ways to break RSA. Also notice that the reason one cannot compute $d$ is partially because one cannot compute $\varphi(n) = (p-1)(q-1)$ from $n$. This is in contrast to the case of primes, where $\varphi(p) = p - 1$.

The remaining training on number theory will be given as we move along.

| CSCI-GA.3210-001<br>MATH-GA.2170-001 | **Introduction to Cryptography** | Feb 15, 2012 |
|---|---|---|

# Lecture 4

*Lecturer: Yevgeniy Dodis* — *Spring 2012*

This lecture will study the notion of *hardcore bit* for a given OWF $f$. Intuitively, such a hardcore bit $h(x)$ is easy to compute from $x$, but almost impossible to even *guess* well from $f(x)$. We will see specific examples of hardcore bits for modular exponentiation, RSA and Rabin's squaring function. Next we will show a groundbreaking result of Goldreich-Levin, that (more or less) shows a general hardcore bit for *any* OWF. We will then consider two natural applications of hardcore bits to the problem of encryption. Firstly, we will show an inuitively good public-key encryption for one bit, and then a "plausible" secret-key encryption which encrypts $k+1$ bits with $k$-bit key (thus beating the Shannon information-theoretic bound). We will then try to extend the hardcore bit construction to extracting many "pseudorandom bits", by analogy to the S/Key system. We will notice that our many-bit construction seems to satisfy a very special notion of security, which we call "next-bit unpredictability". We then will make a formal definition of a *pseudorandom generator*, which seems to be a more relevant primitive for our encryption applications. We stop by asking the question of whether our "next-bit secure" construction is indeed a pseudorandom generator.

## 1 Hardcore Bits

Last lecture we addressed some of the criticism over straightforward usages of OWF's, OWP's and TDP's. Specifically, our main criticism was the fact that a OWF $f(x)$ could reveal a lot of partial information about $x$ (remember generic example $f(x_1, x_2) = (f'(x_1), x_2)$ that reveals half of its input bits, or more realistic one of exponentiation $f(x) = g^x \bmod p$ that reveals the $LSB(x) = y^{(p-1)/2}$).

The obvious solution seemed to try to *completely* hide all information about $x$, given $f(x)$. However, this leads to a vicious circle, since it is really equivalent to the problem of secure encryotion that we started from. Instead, we explore the idea of *completely* hiding not *all*, but only a *specific and carefully chosen partial information about* $x$, when given $f(x)$. The first preliminary step towards this goal is to determine how to completely hide exactly just *one* bit of information about the plaintext $x$. This leads us to the following definition:

Definition 1 [Hardcore bit] A function $h : \{0,1\}^* \to \{0,1\}$ is called a hardcore bit for a function $f$ if

- $h(x)$ is polynomial time computable (from $x$):

$$(\exists \text{ poly-time } H)(\forall x)[H(x) = h(x)]$$

- No PPT algorithm that can predict $h(x)$ given $f(x)$ better than flipping a coin:

$$(\forall \text{ PPT } A)\ \Pr[A(f(x)) = h(x) \mid x \leftarrow^r \{0,1\}^k] \leq \frac{1}{2} + \mathsf{negl}(k)$$

$\diamondsuit$

**Remark 1** *Notice, we compare the success of $A$ to a $\frac{1}{2} + \mathsf{negl}(k)$ rather than $\mathsf{negl}(k)$, as we did for OWF's. The reason is that the output of $h$ is now only one bit, so $A$ can always guess if with probability $\frac{1}{2}$ by flipping a coin. Of course, we could have said that $h$ is difficult to compute by saying that $A$ succeeds with probability at most $1-\varepsilon$, where $\varepsilon$ is non-negigible. However, we really want to say much more (and that is why we started with $h(x)$ being just 1 bit for now): not only is $h$ hard to* compute*, it is even hard to* predict*.*

**Remark 2** *We can also naturally define hardcore bits for* collection *of OWFs, where $h$ can depend on the public key $PK$ of $f$.*

Thus, a hardcore bit $h(x)$ pinpoints an aspect of $x$ that is truly hidden given $f(x)$. Namely, the knowledge of $f(x)$ does not allow us to predict $h(x)$ any better than *without it* (i.e., by flipping a coin), so $h(x)$ looks random given $f(x)$. It is to be noted that we need not require that $h(x)$ be a bit that is selected from the string $x$ itself, but, in general, may depend on $x$ in more complex (but efficiently computable) ways. This is not inconsistent with the idea that $h(x)$ is supposed to represent some general information about $x$. We might want to attempt the construction in two ways:

1. Taking as hypothesis that a *concrete* function is OWF, exhibit a hardcore bit for that function. (This is a useful, but not very general construction.)

2. Taking as hypothesis that an *arbitrary* function is OWF, exhibit a hardcore bit for that function. (This is the strongest construction we can hope for.)

## 2 Hardcore Bits for Concrete OWF's

The concrete function that we consider is the exponentiation function mod $p$, $f(x) = y = g^x \bmod p$, where $g$ is the generator of $\mathbb{Z}_p^*$. Recall, the least significant bit $LSB(x)$ was not hardcore for $f$, since it could be computed from $y$ in polynomial time. Instead, we define the most significant bit of $x$, $MSB$, as follows:

$$MSB(x) = \begin{cases} 0, & \text{if } x < \frac{p-1}{2} \\ 1, & \text{if } x \geq \frac{p-1}{2} \end{cases}$$

**Remark 3** *$MSB(x)$ not defined to be simply $x_1$ in order to make it unbiased, since the prime $p$ is not a perfect power of 2.*

**Theorem 1** *If $f(x) = (g^x \bmod p)$ is a OWP, then $MSB(x)$ is a hardcore bit for $f$.*

**Proof Sketch:** A rigorous proof for the above theorem exists. It is our usual proof by contradiction, which takes as hypothesis that $MSB$ is *not* an hardcore bit for $f$, and proves that $f$ is not OWF. The proof is constructive, and explicitly transforms any PPT that can compute $MSB(x)$ from $f(x)$ with non-negligible advantage, into a PPT that can compute the Discrete Log with non-negligible probability. This proof is, however, somewhat technical, so we settled for a simpler, but quite representative result stated below. □

**Lemma 1** *If there exists a* PPT *that can* always *compute* $MSB(x)$ *from* $f(x)$*, then there is a* PPT *that can* always *invert* $f(x)$ *(i.e., compute the discrete log of* $y = f(x)$*).*

**Proof:** The idea of the algorithm is very simple. We know that $LSB(x) = x_k$ is easy to compute given $y = g^x \bmod p$. This way we determine $x_k$. Now we can transform $y$ into $g^{[x_1 \ldots x_{k-1} 0]} = (g^{[x_1 \ldots x_{k-1}]})^2 \bmod p$ (by dividing $y$ by $g$ if $x_k = 1$). We also know how to extract square roots modulo $p$. So it seems like we can compute $g^{[x_1 \ldots x_{k-1}]}$, and keep going the same way (take $LSB$, extract square root, etc.) until we get all the bits of $x$. However, there is a problem. The problem is that $g^{[x_1 \ldots x_{k-1} 0]}$ has *two* square roots: $y_0 = g^{[x_1 \ldots x_{k-1}]}$ (the one we want) and $y_1 = (-g^{[x_1 \ldots x_{k-1}]}) = g^{\frac{p-1}{2} + [x_1 \ldots x_{k-1}]}$ (here we used the fact that $-1 = g^{(p-1)/2}$). So after we compute the square roots $y_0$ and $y_1$, how do we know which root is really $y_0 = g^{[x_1 \ldots x_{k-1}]}$? Well, this is exactly what the hardcore bit $MSB$ tells us! Namely, $MSB(Dlog(y_0)) = 0$ and $MSB(Dlog(y_1)) = 1$. The complete algorithm follows.

$i = k$;
**while** $(y \geq 1)$ **do** /* $y = f(x)$ */
**begin**
  output ( $x_i = LSB(Dlog(y))$ );
  /* Assertion: $x = [x_1 x_2 \ldots x_i]$ */
  **if** ( $x_i == 1$ ) **then** $y := y/g$;
  /* Assertion: $y = g^{[x_1 x_2 \ldots x_{i-1} 0]} = (g^{[x_1 x_2 \ldots x_{i-1}]})^2$ */
  **Let** $y_0$ and $y_1$ be square roots of $y$;
  **If** ( $MSB((Dlog(y_1)) == 0$ )  /* Using hypothetical algortihm */
    **then** $y := y_1$
    **else** $y := y_2$
  $i := i - 1$;
**end**

To summarize, the value of $MSB$ plays a critical role in distinguishing which square root of $y$ corresponds to $x/2$. This enables us to use the $LSB$ iteratively, so that the process is continued to extract all the bits of $x$. □

It turns out that the other OWF's we study have natural hardcore bits as well:

1. $LSB$ and $MSB$ are hardcore bits for Rabin's Squaring Function.
2. All the bits of $x$ are hardcore bits for $RSA$.

## 3 Construction of a hardcore bit for arbitrary OWF

Looking at the previous examples, we would now like to see if any OWF $f$ has some easy and natural hardcore bits. Specifically, it would be great if at least one the following two statements was true:

1. Any OWF has some particular bit $x_i$ in $x$ ($i$ depends on $f$) which is hardcore for $f$.
2. A concrete boolean function $h$ (which is not necessarily an input bit) is a hardcore bit for *all* OWF's $f$.

Unfortunately, both of these hopes are false in general.

1. From arbitrary OWF $f$, it is possible to construct another OWF $g$, such that none of the bits of $x$ are hardcore for $g$. (cf. Handout).
2. For any boolean function $h$ and OWF $f$, if we let $g(x) = f(x) \circ h(x)$, then: (1) $g$ is also a OWF; (2) $h$ is not hardcore for $g$. Part (1) follows from the fact that an inverter $A$ for $g$ would imply the one for $f$. Indeed, given $y = f(x)$, we can ask the inverter $A$ for $g$ to invert both $f(x) \circ 0$ and $f(x) \circ 1$, and see if at least one of them succeeds. Part (2) is obvious. Thus, no "universal" $h$ exists.

Despite these negative news, it turns out that we nevertheless have a very simple hardcore bit for an arbitrary OWF. This is the celebrated Goldreich Levin construction.

## 4 Goldreich Levin Construction

We will begin with a definition that generalizes the concept of selecting a specific bit from a binary string.

Definition 2 [Parity] If $x = x_1x_2 \dots x_k \in \{0,1\}^k$, and $r = r_1r_2 \dots r_k \in \{0,1\}^k$, then $h(x,r) = r_1x_1 \oplus r_2x_2 \dots \oplus r_kx_k = (r_1x_1 + r_2x_2 + \dots + r_kx_k \bmod 2)$ is called the parity of $x$ with respect to $r$. ♢

Notice, $r$ can be viewed as a selector for the bits of $x$ to be included in the computation of parity. Further, the expression for the notation for the inner product, $\cdot$, can be used profitably, i.e., $h(x,r) = (r \cdot x)$ can be viewed as the inner product of binary vectors $x$ and $r$ modulo 2. The "basis strings" $e_i$ with exactly one bit $r_i = 1$, give the usual specific bit selection $x_i$.

The Goldrecih-Levin theorem essentially says that "if $f$ is a OWF, then most parity functions $h(x,r)$ are hardcore bits for $f$." To make the above statement more precise, it is convenient to introduce an auxiliary function $g_f(x,r) = f(x) \circ r$ (the concatenation of $f(x)$ and $r$), where $|x| = |r|$. Notice, a random input for $g_f$ samples *both* $r$ and $x$ at random from $\{0,1\}^k$, so $h(x,r) = x \cdot r$ indeed computes a "random parity for (randomly selected) $x$". Notice also, that if $f$ is a OWF/OWP/TDP, then so is $g_f$ (in particular, $g_f$ is a permutation if $f$ is). Now, the Goldrecih-Levin theorem states that

**Theorem 2 (Goldreich-Levin Bit)** *$f$ is a* OWF*, then $h(x,r)$ is a hardcore bit for $g_f$. More formally:*

$$(\forall \text{ PPT } A) \quad \Pr[A(f(x),r) = (x \cdot r) \mid x, r \leftarrow^r \{0,1\}^k] < \frac{1}{2} + \mathsf{negl}(k)$$

**Remark 4** *A hardcore bit for $g_f$ is as useful as the one for $f$, since $f$ is really computationally equivalent to $g_f$ (since $r$ is part of the output, inverting $g_f$ exactly reduces to inverting $f$). Thus, we will often abuse the terminology and say "since $f$ is a* OWF*, let us take its hardcore bit $h(x)$". But that we really mean that in case we are not aware of some simple hardcore bit for a specific $f$, we can always take the Goldreich-Levin bit for $g_f$ and use it instead. Similar parsing should be given to statement of the form "every* OWF *has a hardcore bit". Again, in the worst case always use the Goldrecih-Levin bit for $g_f$. Finally, another popular interpretation of this result is that "most parities of $f$ are hardcore".*

A rigorous proof of the Goldreich-Levin theorem exists. For simplicity, we will assume that $f$ (and thus $g_f$) are *permutations*, so that $(x \cdot r)$ is uniquely defined given $f(x) \circ r$.

The proof proceeds proof by contradiction, by taking as hypothesis that $h(x,r)$ is *not* a hardcore bit for $g_f$, and proves that $f$ is not a OWF. The proof is constructive, and explicitly transforms any PPT $A$ that can compute $h(x,r)$ with non negligible probability for most values of $r$, given $f(x)$ and $r$, into a PPT $B$ that can compute $x$ with non negligible probability, given $f(x)$. However and despite the simplicity of theorem statement, the full proof is extremely technical. Therefore, we will again only give a good intuition of why it works.

Before going to the general proof, in the next subsection we give two simple cases which make the proof considerably simpler. A reader not interesting in the technical proof is adviced to read only these two cases, and skip the following subsection describing the general case.

### 4.1 Simple Cases

First, assume that we are given a PPT $A$ that *always* computes $(x \cdot r)$ given $f(r) \circ r$. Well, then everything is extremely simple. Given $y = f(x)$ that we need to invert, we already observed that $x \cdot e_i = x_i$ is the $i$-th bit of $x$, where $e_i$ is a vector with 1 only at poistion $i$. Thus, asking $A(y, e_i)$ will give us $x_i$, so we can perfectly learn $x$ bit by bit.

Unfortunately, our assumption on $A$ is too strong. In reality we only know that it succeeds on a slight majority of $r$'s. In particular, maybe it always refuses to work for "basis" $r = e_i$. So let us be more reasonable and assume that

$$\Pr[A(f(x),r) = (x \cdot r) \mid x, r \leftarrow \{0,1\}^k] > \frac{3}{4} + \varepsilon \tag{1}$$

where $\varepsilon$ is non-negligible. Here we use $3/4$ instead of $1/2$ for the reason to be clear in a second. But first we need a definition. Given a *particular* value $x$, we let

$$Succ(x) \stackrel{\text{def}}{=} \Pr[A(f(x),r) = (x \cdot r) \mid r \leftarrow \{0,1\}^k] \tag{2}$$

Namely, $Succ(x)$ is "how well" $A$ predicts $x \cdot r$ for a particular $x$ (averaged over $r$). Then, Equation (1) is equivalent to saying that the *expected value* of $Succ(x)$ is non-trivially greater than 3/4:

$$\mathbb{E}[Succ(x)] > \frac{3}{4} + \varepsilon \tag{3}$$

Let us call $x$ "good" if $Succ(x) > \frac{3}{4} + \frac{\varepsilon}{2}$. Then a simple averaging argument show that

$$\Pr[x \text{ is good} \mid r \leftarrow \{0,1\}^k] > \frac{\varepsilon}{2} \tag{4}$$

Indeed, if Equation (4) is false, then conditioning on whether or not $x$ is good, the largest that $\mathbb{E}[Succ(x)]$ can be is

$$\mathbb{E}[Succ(x)] \leq \frac{\varepsilon}{2} \cdot 1 + \left(1 - \frac{\varepsilon}{2}\right) \cdot \left(\frac{3}{4} + \frac{\varepsilon}{2}\right) < \frac{3}{4} + \varepsilon$$

which is contradicting Equation (3).

We now will construct an inverter $B$ for $f$ which will only work well for good $x$. But since the fraction of good $x$ is non-negligible (at least $\varepsilon/2$ by Equation (4)), the existence of $B$ will contradict the fact that $f$ is a $\mathsf{OWF}$. Thus, in the following we will assume that $x$ is good when analyzing the success of $B$.

The idea is to notice that for any $r \in \{0,1\}^k$

$$(x \cdot r) \oplus (x \cdot (r \oplus e_i)) = \left(\sum_{j \neq i} r_j x_j + r_i x_i \bmod 2\right) \oplus \left(\sum_{j \neq i} r_j x_j + (1 - r_i) x_i \bmod 2\right) = x_i$$

Moreover both $r$ and $(r \oplus e_i)$ are *individually random* when $r$ is chosen at random. Hence, for any fixed index $i$ and any good $x$,

$$\begin{aligned} \Pr[A(y,r) \neq (x \cdot r) \mid y = f(x),\ r \leftarrow \{0,1\}^k] &< \frac{1}{4} - \frac{\varepsilon}{2} \\ \Pr[A(y, r \oplus e_i) \neq (x \cdot (r \oplus e_i)) \mid y = f(x),\ r \leftarrow \{0,1\}^k] &< \frac{1}{4} - \frac{\varepsilon}{2} \end{aligned}$$

Thus, with probability at least $1 - 2(\frac{1}{4} - \frac{\varepsilon}{2}) = \frac{1}{2} + \varepsilon$, $A$ will be correct in *both* cases, and hence we correctly recover $x_i$ with probability $\frac{1}{2} + 2\varepsilon$. Thus, using $A$ we can have a $\mathsf{PPT}$ procedure $B'(y,i)$ (which is part of "full" $B$ below) which will sample a random $r$ and return $A(y,r) \oplus A(y \oplus e_i)$, such that for *any* good $x$ and any $i$,

$$\Pr[B'(y,i) = x_i \mid y = f(x)] \geq \frac{1}{2} + \varepsilon \tag{5}$$

Namely, we can predict any particular bit $x_i$ with probability greater than 1/2. Thus, repeating $B'(y,i)$ roughly $t = O(\log k/\varepsilon^2)$ times (each time picking a brand new $r$; notice also that $t$ is polynomial in $k$ by assumption on $\varepsilon$) and taking the majority of the answers, we determine each $x_i$ correctly with probability $1 - 1/k^2$.[1] Namely, by taking the majority

[1] This follows from the Chernoff's bound, stating that the probability the an average of $t$ independent experiments is "$\varepsilon$-far" from its expectation is of the order $e^{-\Omega(t\varepsilon^2)}$. More formally, let $Z_j$ be the indicator variable which is 1 if the $j$-test was correct, where $j = 1 \ldots t$. Then all $Z_j$ are independent and $\mathbb{E}[Z_j] \geq \frac{1}{2} + \varepsilon$. Then, if $Z = \sum_j Z_j$, we have $\mathbb{E}[Z] \geq t(\frac{1}{2} + \varepsilon)$, and $\Pr[Z < t/2] \leq e^{\Omega(t\varepsilon^2)}$ by the Chernoff's bound.

vote we essentially "amplified" the success of $B'$ and obtained an algortihm $B''(y,i)$ such that for *any* good $x$ and any $i$,

$$\Pr[B''(y,i) = x_i \mid y = f(x)] \geq 1 - \frac{1}{k^2} \tag{6}$$

We now repeat $B''$ for all indices $i$, therefore defining $B(y)$ as running $B''(y,1)\ldots B''(y,k)$. We get that the probability that *at least one* $x_i$ *is wrong* is at most $k/k^2 = 1/k$, so $B$ recovers the entire (good) $x$ correctly with probability $1 - 1/k$, which is certainly non-negligible, contradicting the one-wayness of $f$.

### 4.2 General Case* (Technical, can be skipped)

Still, our assumption about the success of $A$ with probability $\frac{3}{4} + \varepsilon$ is too much. We can only assume $\frac{1}{2} + \varepsilon$. It turns out the proof follows the same structure as the simplistic proof above, except the algorithm $B''$ will be defined more carefully.

Specifically, recall our assumption now is that

$$\Pr[A(f(x), r) = (x \cdot r) \mid x, r \leftarrow \{0,1\}^k] > \frac{1}{2} + \varepsilon \tag{7}$$

where $\varepsilon$ is non-negligible. As earlier, Equation (7) is equivalent to saying that the *expected value* of $Succ(x)$ (defined as in the previous section) is non-trivially greater than 1/2:

$$\mathbb{E}[Succ(x)] > \frac{1}{2} + \varepsilon \tag{8}$$

Similarly to the previous section, we call $x$ "good" if $Succ(x) > \frac{1}{2} + \frac{\varepsilon}{2}$. Then the same averaging argument as earlier implies that

$$\Pr[x \text{ is good} \mid r \leftarrow \{0,1\}^k] > \frac{\varepsilon}{2} \tag{9}$$

Thus, as in the previous case, it suffices to show how to invert good $x$, except the definition of "good" is considerably weaker than before: $Succ(x) > \frac{1}{2} + \frac{\varepsilon}{2}$ instead of a much more generous $Succ(x) > \frac{3}{4} + \frac{\varepsilon}{2}$.

As earlier, though, the way we construct our inverter $B$ is by constructing a "bit predictor" $B''(y,i)$ which satisfies Equation (6), except "good" $x$ is defined differently: for *any* good $x$ and any $i$,

$$\Pr[B''(y,i) = x_i \mid y = f(x)] \geq 1 - \frac{1}{k^2} \tag{10}$$

Then $B$ is defined as before to be $B''(y,1)\ldots B''(y,k)$. Hence, we "only" need to define a new, more sophisticated $B''$ satisfying Equation (10).

The definition and the analysis of $B''$ form the heart of the Goldreich-Levin proof. The idea is the following. Recall, the algortihm $B''$ from the previous section was defined as follows: for some parameter $t$, $B''$ chose $t$ random values $r_1 \ldots r_t \in \{0,1\}^k$, and then output the majority of values of $A(y, r_j) \oplus A(y, r_j \oplus e_i)$, where $j$ randges from 1 to $t$. The problem is that we can no longer argue that $\Pr_r[A(y,r) \oplus A(y, r \oplus e_i) = x_i] \geq \frac{1}{2} + \varepsilon$. A *wrong* argument to get a weaker, but still sufficient bound would be to say each value is correct

with probability at least $\frac{1}{2} + \frac{\varepsilon}{2}$ , so we succeed if either both are right or wrong, which happens with probability at least

$$\left(\frac{1}{2} + \frac{\varepsilon}{2}\right)^2 + \left(\frac{1}{2} - \frac{\varepsilon}{2}\right)^2 \geq \frac{1}{2} + \frac{\varepsilon^2}{2}$$

The problem, of course, is that the success of $A$ on *correlated* values $r$ and $r \oplus e_i$ is not independent, and so we cannot just multiply these probabilities.

Instead, we have to built a more sophisticated predictor $B''$. As before $B''$ will choose $t$ random values $r_1, \ldots, r_t \in \{0,1\}^k$ (where we will determine $t$ shortly). Now, however, $B''$ will also *guess* the correct value for $(x \cdot r_i)$. Specifically, $B''$ will choose $t$ random bits $b_1, \ldots, b_t \in \{0,1\}$, and will only work correctly, as explained below, if $x \cdot r_j = b_j$ for *all* $j = 1 \ldots t$. This immediately loses a factor $2^{-t}$ in the success probability of $B''$, so we cannot make $t$ too large. Technically, this also means that we will satisfy Equation (5) only conditioned on the event $E$ stating that all the $t$ guesses of $B''$ are correct: for *any* good $x$ and any $i$,

$$\Pr[B''(y,i) = x_i \mid y = f(x), E \text{ is true}] \geq 1 - \frac{1}{k^2} \tag{11}$$

Luckily, this still suffices to prove our result if $2^{-t}$ is non-negligible (which it will be), since then

$$\Pr[B(y) = x \mid y = f(x)] \geq \Pr(E) \cdot \Pr[B(y) = x \mid y = f(x), E \text{ is true}] \geq 2^{-t} \cdot \left(1 - \frac{1}{k}\right)$$

The point, however, is that if $B''$ correctly guessed all the $t$ paritites $x \cdot r_j$ (which we assume for now), then $B''$ also correctly knows $2^t$ parities of all the linear combinations of the $r_j$'s. Concretely, for any non-empty subset $J \subseteq \{1 \ldots t\}$, we let $r_J = \oplus_{j \in J}\ r_j$ and $b_J = \oplus_{j \in J}\ b_j$. Then

$$x \cdot r_J = x \cdot (\oplus_{j \in J}\ r_j) = \oplus_{j \in J}\ (x \cdot r_j) = \oplus_{j \in J}\ b_j = b_J$$

Now we will let $B''$ call $A$ on $2^t$ values $A(y, r_J \oplus e_i)$, for all non-empty subsets $J \subseteq \{1 \ldots t\}$, and output the majority of $b_J \oplus A(y, r_J \oplus e_i)$. The rational is that whenever $A$ is correct on $r_J \oplus e_i$, the value

$$b_J \oplus A(y, r_J \oplus e_i) = x \cdot r_J \oplus x \cdot (r_J \oplus e_i) = x \cdot e_i = x_i,$$

as needed. The tricky part is to argue that for "small enough" $t$, $B''$ is correct with probability $1 - 1/k^2$ (once again, conditioned on $E$ being true).

We do this as follows. Define an indicator random variable $Z_J$ to be 1 if and only if $A(y, r_J \oplus e_i) = x \cdot (r_J \oplus e_i)$; i.e., if $A$ is correct on $r_J \oplus e_i$. Then, $B''$ is incorrect if and only if a majority of $Z_J$ are incorrect, i.e. $Z \stackrel{\text{def}}{=} \sum_{J \neq \emptyset} Z_J < 2^{t-1}$. But let us compute the expected value of $Z$. First, for any non-empty $J$, $r_J$ is random in $\{0,1\}^k$, and thus, $r_J \oplus e_i$ is also random. Since $x$ is good, this means $\mathbb{E}[Z_J] \geq \frac{1}{2} + \frac{\varepsilon}{2}$, so $\mathbb{E}[Z] \geq 2^{t-1}(1+\varepsilon)$. On the other hand, the probability that $B''$ failed is $\Pr[Z < 2^{t-1}]$.

Ideally, we would like to use the Chernoff's bound, like we did before, but we cannot do it, since the values $Z_J$ are not independent. Luckily, there are *pairwise independent.*

This means than for any non-empty and distinct subsets $I$ and $J$, the values $r_I$ and $r_J$ are independent, which means that $r_I \oplus e_i$ and $r_J \oplus e_i$ are independent, which in turn means that $Z_I$ and $Z_j$ are independent. For such pairwise independent random variables, it turns out we can apply the so called Chebyschev's inequality which states that

**Lemma 2 (Chebyschev's inequality)** *If $W = \sum_{j=1}^{T} W_j$, where $W_j$ are pairwise independent indicator variables with mean at least $p$, then for any $\delta > 0$,*

$$\Pr[Z < T(p-\delta)] \leq \frac{1}{\delta^2 T}$$

We now apply this lemma to our $T = 2^t - 1$ variables $Z_J$ by writing

$$\Pr[Z < 2^{t-1}] \leq \Pr\left[Z \leq (2^t-1)\left(\left(\frac{1}{2}+\frac{\varepsilon}{2}\right)-\frac{\varepsilon}{2}\right)\right] \leq \frac{4}{(2^t-1)\varepsilon^2}$$

By setting $t = \log(1+4k^2/\varepsilon^2) = O(\log(k/\varepsilon))$, we get $\Pr[B''(y,i) \neq x_i \mid E \text{ is true}] \leq 1/k^2$, as needed. The only thing to observe is that $2^{-t} = O(\varepsilon^2/k^2)$ is indeed non-negligible, since $\varepsilon$ is non-negligible.

This concludes the proof of the Goldreich-Levin's Theorem.

## 5 Public Key Cryptosystem for One Bit

We will now look at how to make use of what we have at hand. We will not be terribly rigorous for the time being, and will proceed with the understanding that speculative adventures are acceptable. This time we will hand-wave a little, but will return to the problematic sections in the next lecture.

It seems intuitive that, if we have a hardcore bit, we *should* be able to send *one* bit of information with complete security in the Public Key setting. And we show exactly that.

- **Scenario**

  Bob($B$) wants to send a bit $b$ to Alice($A$). Eve($E$) tries to get $b$. Alice has a public key $PK$ and a secret key $SK$ hidden from everybody.

- **Required Primitives**

  1. TDP $f$ will be the public key $PK$ and its trapdoor information $t$ will be Alice's secret key $SK$.
  2. Hardcore bit $h$ for $f$. If needed, can apply Goldrecih-Levin to get it.

- **Protocol**

  $B$ selects a random $x \in \{0,1\}^k$ and sends $A$ the ciphertext $c = \langle f(x), h(x) \oplus b \rangle$.

- **Knowledge of the Concerned Parties before Decryption**

  $B$: $b, x, c, f, h$.

  $E$: $c, f, h$.

  $A$: $c, t, f, h$.

- **Decryption by A**

  $x$ is obtained from $f(x)$ using the trapdoor $t$;

  $h(x)$ is computed from $x$;

  $b$ is obtained from $(h(x) \oplus b)$ using $h(x)$.

- **Security from E**

  Intuitively, to learn anything about $b$, $E$ must learn something about $h(x)$. But $E$ only knows $f(x)$. Since $h$ is hardcore, $E$ cannot predict $h(x)$ better than flipping a coin, so $b$ is completely hidden.

We note that this scheme is grossly inefficient. Even though it *seems* like we are using an elephant to kill an ant, look what we accoplished: we constructed the first secure public-key cryptosystem!

Having said this, we would still like to improve the efficiency of this scheme. Can we send arbitrary number of bit by using a single $x$ above? More generally, can we extract a lot of random looking bits from a single $x$?

## 6 Public Key Cryptosystem for Arbitrary Number of Bits

We will try to generalize the system in a manner analogous to what we did with the S/Key system. Remember, there we published the value $y_0 = f^T(x)$, and kept giving the server the successive preimages of $y_0$. So maybe we can do the same thing here, except we will use *hardcore bits of successive preimages to make them into a good one-time pad!* Notice also that publishing $f^t(x)$ will still allow Alice to get back all the way to $x$ since she has the trapdoor.

- **Scenario**

  Bob($B$) wants to send a string $m = m_1 \dots m_n$ to Alice($A$). Eve($E$) tries to get "some information" about $m$. Alice has a public key $PK$ and a secret key $SK$ hidden from everybody.

- **Required Primitives**

  1. As before, $\mathsf{TDP}$ $f$ will be the public key $PK$ and its trapdoor information $t$ will be Alice's secret key $SK$.
  2. Hardcore bit $h$ for $f$. If needed, can apply Goldrecih-Levin to get it.

- **Protocol**

  $B$ selects a random $x \in \{0,1\}^k$ and sends $A$ the ciphertext $c = \langle f^n(x), G'(x) \oplus m \rangle$, where

  $$G'(x) = h(f^{n-1}(x)) \circ h(f^{n-2}(x)) \circ \dots \circ h(x) \tag{12}$$

  Notice, $G'(x)$ really serves as a "computational one-time pad".

- **Knowledge of the Concerned Parties before Decryption**

  $B$: $m, x, c, f, h$.

  $E$: $c, f, h$.

  $A$: $c, t, f, h$.

- **Decryption by A**

  $x$ is obtained from $f^n(x)$ using the trapdoor $t$ by going "backwards" $n$ times;

  $G'(x)$ is computed from $x$ by using $h$ and $f$ in the "forward" direction $n$ times;

  $m$ is obtained from $(G'(x) \oplus m)$ using $G'(x)$.

- **Security from E**

  The intuition about the security is no longer that straightforward. Intuitively, if we let $p_1 = h(f^{n-1}(x)), \ldots, p_n = h(x)$ be the one-time pad bits, it seems like $p_1$ looks random given $f^n(x)$, so $m_1$ is secure for now. On the other hand, $p_2 = h(f^{n-2}(x))$ looks secure even given $f^n(x)$ and $p_1$ (since both can be computed from $f^{n-1}(x)$ and $p_2$ is secure even if $f^{n-1}(x)$ is completely known. And so on. So we get a very strange kind of "security". Even if the adversary knows $f^n(x)$ (which he does as it is part of $c$), and even if he somehow learns $m_1, \ldots, m_{i-1}$, which would give it $p_1, \ldots, p_{i-1}$, he still cannot predict $p_i$, and therefore, $m_i$ is still secure. So we get this "next-bit security": given first $(i-1)$ bits, $E$ cannot predict the $i$-th one. It is completely unclear if

  - "Next-bit security" is what we really want from a good encryption (we certainly want as least this security, but does suffice?) For example, does our system satisfy analogously defined "previous-bit security"?
  - Our scheme satisfies some more "reasonable" notion of security.

  The answers to these questions will come soon.

At this point, we turn our attention to Secret Key Cryptography based on symmetric keys. We would like to contemplate whether we could transcend Shannon's Theorem on key lengths.

# 7 Secret Key Cryptosystems

Recall, our main question in secret key encryption was to break the Shannon bound. Namely, we would like to encrypt a message of length $n$ using a secret key of a much smaller size $k$, i.e. $k < n$ (and hopefully, $k \ll n$). We right away propose a possible solution by looking at the corresponding public-key example for encrypting many bits.

Recall, in the public key setting we used $G'(x)$ (see Equation (12)) as a "computational one-time pad" for $m$, where $x$ was chosen at random by Bob. Well, now we can do the same thing, but make $x$ the shared secret! Notice also that now we no longer need the trapdoor, so making $f$ OWP suffices.

- **Scenario**

  Bob($B$) wants to send a string $m = m_1 \dots m_n$ to Alice($A$). Eve($E$) tries to get "some information" about $m$. Alice and Bob share a random $k$-bit key $x$ which is hidden from Eve.

- **Required Primitives**

  1. OWP $f$ which is known to everyone.
  2. Hardcore bit $h$ for $f$. If needed, can apply Goldrecih-Levin to get it.

- **Protocol**

  $B$ sends $A$ the ciphertext $c = G'(x) \oplus m$, where $G'(x)$ is same as in Equation (12).

  $$G'(x) = h(f^{n-1}(x)) \circ h(f^{n-2}(x)) \circ \dots \circ h(x) \tag{13}$$

  As before, $G'(x)$ really serves as a "computational one-time pad".

- **Knowledge of the Concerned Parties before Decryption**

  $B$: $m, x, c, f, h$.

  $E$: $c, f, h$.

  $A$: $c, x, f, h$.

- **Decryption by A**

  $G'(x)$ is computed from secret key $x$ by using $h$ and $f$ in the "forward" direction $n$ times;

  $m$ is obtained from $(G'(x) \oplus m)$ using $G'(x)$.

- **Security from E**

  Again, the intuition about the security is not straightforward. Similar to the public-key example, it seems like we get what we called "next-bit security": given first $(i-1)$ bits of $m$ (or of $G'(x)$), $E$ cannot predict the $i$-th bit of $m$ (or $G'(x)$). It is completely unclear if

  – "Next-bit security" is what we really want from a good encryption (we certainly want as least this security, but does suffice?) For example, does our system satisfy analogously defined "previous-bit security"?

  – Our scheme satisfies some more "reasonable" notion of security.

  Again, the answers to these questions will come soon,

Before moving on, we also make several more observations. First, notice that we could make $n$ very large (in particular, much larger than $k$). Also, we can make the following optimization. Recall that in the public key scenario we also send $f^n(x)$ to Alice so that she can recover $x$. Now, it seems like there is no natural way to use it, so we really computed

it almost for nothing... But wait, we could use $f^n(x)$ to make our one-time pad longer! Namely, define

$$G(x) = f^n(x) \circ G'(x) = f^n(x) \circ h(f^{n-1}(x)) \circ h(f^{n-2}(x)) \circ \ldots \circ h(x) \tag{14}$$

Now we can use $G(x)$ as the one-time pad for messages of length $n + k$, which is *always* greater than our key size $k$, even if $n = 1$! Indeed, we claim that our intuitively defined "next-bit security" holds for $G(x)$ as well. Indeed, for $i < k$, $f^n(x)$ is *completely random* (since $x$ is random), so predicting the $i$-th bit based on the first $(i - 1)$ bits is hopeless. While for $i > k$ our informal argument anyway assumed that Eve knows $f^n(x)$ (it was part of the encryption). We will discuss later if it really pays off in the long run to use this efficiency improvement (can you think of a reason why it might be good to keep the same $x$ for encrypting more than one message?)

However, using either $G(x)$ or $G'(x)$ as our one-time pads still has its problems that we mentioned above. Intuitively, what we really want from a "computational one-time pad" is that it really *looks completely random to Eve*. We now formalize what it means, by defining an extremely important concept of a *pseudorandom number generator*.

# 8 Pseudo Random Generators

Intuitively, a *pseudorandom number generator* (PRG) stretches a short random seed $x \in \{0,1\}^k$ into a longer output $G(x)$ of length $p(k) > k$ which nevertheless "looks" like a random $p(k)$-bit strings to any computationally bounded adversary. For clear reasons, the adversary here is called a *distinguisher*.

Definition 3 [Pseudo Random Generator] A deterministic polynomial-time computable function $G : \{0,1\}^k \to \{0,1\}^{p(k)}$ (defined for all $k > 0$) is called a *pseudorandom number generator* (PRG) if

1. $p(k) > k$ (it should be stretching).
2. There exists no PPT distinguishing algorithm $D$ which can tell $G(x)$ apart from a truly random string $R \in \{0,1\}^{p(k)}$. To define this formally, let 1 encode "pseudorandom" and 0 encode "random". Now we say that for any PPT $D$

$$|\Pr(\ D(G(x)) = 1 \mid x \leftarrow^r \{0,1\}^k\ ) - \Pr(\ D(R) = 1 \mid R \leftarrow^r \{0,1\}^{p(k)}\ )| < \mathsf{negl}(k)$$

♢

We observe that we require the length of $x$ be less than the output length of $G(x)$. This is done since otherwise an identity function will be a trivial (and useless) PRG. It should not be that easy! On the other hand, requiring $p(k) > k$ makes this cryptographic primitive quite non-primitive to construct (no pun intended).

Secondly, we are not creating pseudorandomness from the thin air. We are taking a *truly random seed* $x$, and stretch it to "computationally random" output $G(x)$. In other words, $G(x)$ is computationally indistinguishable from a random sequence (i.e., looks random), only provided that (much shorter seed) $x$ is random.

## 9 Points to Ponder

We would like to conclude with some questions which are food for thought.

1. Is our public-key encryption really good?
2. What about the secret-key encryption?
3. Are $G'(x)$ and $G(x)$ (see Equations (13) and (14)) pseudorandom generators?
4. Can we output the bits of $G'(x)$ in "forward" order?
5. Is "next-bit security" enough to imply a true PRG?

| CSCI-GA.3210-001<br>MATH-GA.2170-001 | **Introduction to Cryptography** Ferbruary 22, 2012 |
|---|---|

# Lecture 5

*Lecturer: Yevgeniy Dodis* *Spring 2012*

In this lecture we formalize our understanding of *next-bit security* and its relationship to pseudorandomness. Namely, we prove that next-bit security implies a PRG. On our way to this proof we introduce the important cryptographic idea of *computational indistinguishability* and the related technique of the *hybrid argument*. Having proved that functions $G(x)$ and $G'(x)$ (which we introduced in the last lecture and which are defined here in Equations (1) and (2)) are next-bit secure and therefor PRG's, we show that Equation (1) can be used to construct a PRG without our having to decide a length in advance. We look at two specific examples of such PRG's: the Blum-Micali generator and the Blum-Blum-Shub generator. Next we examine the relationship between PRG's and OWF's and come to the startling conclusion that asserting the existence of one of these primitives is equivalent to asserting the existence of the other. Finally we introduce the important idea of *forward security* for a PRG and discuss the role of PRG's in real life.

## 1 Next-Bit Unpredictability and PRG's

Last lecture we used the concept of *hardcore bits* to construct public- and secret-key cryptosystems whose security was plausible but unclear. Both of our constructions used a OWP (possibly trapdoor) $f$ and its hardcore bit $h$ and considered iterating $f(x)$ (for a random $x \in \{0,1\}^k$) $n$ times, and output the hardcore bits of $f^i(x)$ in reverse order. It particular, we considered two functions $G' : \{0,1\}^k \to \{0,1\}^n$ and $G : \{0,1\}^k \to \{0,1\}^{k+n}$ defined as

$$G'(x) = h(f^{n-1}(x)) \circ h(f^{n-2}(x)) \circ \ldots \circ h(x) \quad (1)$$

$$G(x) = f^n(x) \circ G'(x) = f^n(x) \circ h(f^{n-1}(x)) \circ h(f^{n-2}(x)) \circ \ldots \circ h(x) \quad (2)$$

Intuitively and by the analogy with the S/Key system, these functions seem to satisfy the following notion of security which we now define formally.

Definition 1 [Next-bit Unpredictability] A deterministic polynomial-time computable function $G : \{0,1\}^k \to \{0,1\}^{p(k)}$ (defined for all $k > 0$) satisfies the *next-bit unpredictability* property if for every index $0 \le i \le p(k)$ and every PPT next-bit predictor $P$

$$\Pr\left(b = g_i \;\middle|\; x \leftarrow^r \{0,1\}^k,\ g = g_1 \ldots g_{p(k)} = G(x),\ b \leftarrow P(g_1 \ldots g_{i-1})\right) < \frac{1}{2} + \mathsf{negl}(k)$$

Namely, no predictor $P$ can succeed in guessing $g_i$ from $G_{i-1} = g_1 \ldots g_{i-1}$ significantly better than by flipping a coin. ♢

We now formally show that $G(x)$ (and hence $G'(x)$ as well) satisfies this property.

**Lemma 1** *If $f$ is a OWP, $h$ is a hardcore bit of $f$ and $G$ is defined by Equation (2), then $G$ is next-bit unpredictable.*

**Proof:** The proof is almost identical to the one we used for the S/Key system. So assume for some $i$ and some PPT $P$ our function $G$ is not next-bit unpredictable. We notice that we must have $i > k$, since otherwise $g_i$ is part of a truly random $f^n(x)$ (remember, $x$ is random), and is independent of $G_i = g_1 \dots g_{i-1}$. Thus, assume $i = k+j$ where $j > 0$. Thus,

$$\Pr\left(P(f^n(x), h(f^{n-1}(x)), \dots, h(f^{n-j+1}(x))) = h(f^{n-j}(x)\right) > \frac{1}{2} + \epsilon$$

We now construct a predictor $A$ which will compute the hardcore bit $h(\tilde{x})$ from $\tilde{y} = f(\tilde{x})$. As with S/key, $A$ simply outputs

$$P(f^{j-1}(\tilde{y}), h(f^{j-2}(\tilde{y})), \dots, h(\tilde{y}))$$

with the hope that $P$ computes $h(f^{-1}(\tilde{y})) = h(\tilde{x})$. The analysis that the advantage of $A$ is $\epsilon$ is the same as with the S/key example, and uses the fact that $f$ is a permutation. □

Having formally verified this, we ask the same question as before. Do $G(x)$ and $G'(x)$ in Equation (1) and Equation (2) really satisfy their purpose of being "computational one-time pads"? Last lecture we also intuitively argued that this means that $G(x)$ (resp. $G'(x)$) should really look indistingushable from a truly random string of length $n + k$ (resp. $n$). We then formalized this property by defining the notion of a *pseudo-random generator*.

DEFINITION 2 [Pseudorandom Generator] A deterministic polynomial-time computable function $G : \{0,1\}^k \to \{0,1\}^{p(k)}$ (defined for all $k > 0$) is called a *pseudorandom number generator* (PRG) if

1. $p(k) > k$ (it should be stretching).
2. There exists no PPT distinguishing algorithm $D$ which can tell $G(x)$ apart from a truly random string $R \in \{0,1\}^{p(k)}$. To define this formally, let 1 encode "pseudorandom" and 0 encode "random". Now we say that for any PPT $D$

$$|\Pr(\ D(G(x)) = 1 \mid x \leftarrow^r \{0,1\}^k\ ) - \Pr(\ D(R) = 1 \mid R \leftarrow^r \{0,1\}^{p(k)}\ )| < \mathsf{negl}(k)$$

♢

Thus, a *pseudorandom number generator* (PRG) stretches a short random seed $x \in \{0,1\}^k$ into a longer output $G(x)$ of length $p(k) > k$ which nevertheless "looks" like a random $p(k)$-bit strings to any computationally bounded adversary. For clear reasons, we call the adversary $D$ a *distinguisher*.

Now, rather than verifying directly if our $G$ and $G'$ are PRG's, we will prove a much more suprising result. Namely, we show that *any* $G$ which satisfies next-bit unpredictability is a PRG.

**Theorem 1** *If an arbitrary $G : \{0,1\}^k \to \{0,1\}^{p(k)}$ is next-bit unpredictable, then $G$ is a* PRG. *More quantitively, if some* PPT *distinguisher for $G$ has advantage $\epsilon$ is telling $G(x)$ apart from a random $R$, then for some index $1 \le i \le p(k)$ there is a* PPT *predictor for $G$ which has advantage at least $\epsilon/p(k)$.*

We notice that the converse (PRG implies next-bit unpredictability) is obvious. Indeed, if some $P$ breaks next-bit unpredcitability of some $G$ at some index $i$, here is a distinguisher $D(y_1 \dots y_{p(k)})$:

Let $g = P(y_1 \dots y_{i-1})$.
If $g = y_i$ output 1 ("pseudorandom"), esle output 0 ("random")

Indeed, by assumption, if $y = G(x)$, then $\Pr(D(y) = 1) \geq \frac{1}{2} + \epsilon$. On a random string $y = R$, clearly $\Pr(D(y) = 1) = \frac{1}{2}$, since there is no way $P(R_1 \dots R_{i-1})$ can predict a totally fresh and independent $R_i$.

We give the proof of Theorem 1 in Section 3. The proof uses an extremely important technique called a *hybrid argument*. However, it is a somewhat technical to understand right away. Therefore, we step aside and introduce several very important concepts that will (1) make the proof of Theorem 1 less mysterious; (2) explain better the definition of a PRG by introducing the general paradigm of *computational indistiguishability*; (3) make new definitions similar to that of a PRG very easy to express and unserstand; and (4) introduce the hybrid argument in its generality. We will return to our mainstream very shortly.

## 2 Computational Indistinguishability + Hybrid Argument

The definition of OWF's/OWP's/TDP's had the flavor that

"something is hard to *compute* precisely"

We saw that this alone is not sufficient for cryptographic applications. The definition of a hardcore bit and subsequently of the next-bit unpredictability were the first ones that said that

"something is hard to *predict* better than guessing"

Finally, the definition of a PRG took a next crycial step by saying that

"something is *computationally indistinguishable* from being random"

Not surprisingly, we will see many more cryptographic concepts of a similar flavor where more generally

"something is *computationally indistinguishable* from something else"

Intuitively, "something" will often be the cryptographic primitive we are considering, while "something else" is the ideal (and impossible to achieve/very expensive to compute) object we are trying to efficiently approximate. In order to save time in the future and to understand this concept better, we treat this paradigm in more detail.

Definition 3 Let $k$ be the security parameter and $X = \{X^k\}, Y = \{Y^k\}$ be two ensembles of probability distributions where the descrytpion of $X^k$ and $Y^k$ are of polynomial length in $k$. We say that $X$ and $Y$ are *computationally indistinguishable*, denoted $X \approx Y$, if for any PPT algorithm $D$ (called the *distinguisher*) we have that

$$|\Pr(D(X^k) = 1) - \Pr(D(Y^k) = 1)| < \mathsf{negl}(k)$$

where the probability is taken over the coin tosses of $D$ and the random choices of $X^k$ and $Y^k$. The absolute value above is called the *advantage* of $D$ in distinguishing $X$ from $Y$, denoted $\mathsf{Adv}_D(X,Y)$. $\diamondsuit$

Notice that in this terminology the definition of a $\mathsf{PRG}$ $G : \{0,1\}^k \to \{0,1\}^{p(k)}$ reduces simply to saying that for a random $x \in \{0,1\}^k$ and $R \in \{0,1\}^{p(k)}$, we have

$$G(x) \approx R$$

We give very simple properties of computational indistinguishability.

**Lemma 2** *If $X \approx Y$ and $g$ is polynomial time computable, then $g(X) \approx g(Y)$.*

**Proof:** Assuming a $\mathsf{PPT}$ distibguisher $D$ for $g(X)$ and $g(Y)$, a $\mathsf{PPT}$ distinguisher $D'(z)$ for $X$ and $Y$ simply runs $D(g(z))$, which it can do since $g$ is poly-time. Clearly, $\mathsf{Adv}_{D'}(X,Y) = \mathsf{Adv}_D(g(X), g(Y))$. $\square$

The next result, despite its simplicity is a foundation of a very powerful technique.

**Lemma 3** *If $X \approx Y$ and $Y \approx Z$, then $X \approx Z$. More generally, if $n$ is polynomial in $k$ and $X_0 \approx X_1$, $X_1 \approx X_2$, ..., $X_{n-1} \approx X_n$, then $X_0 \approx X_n$. More quantitively, if some distinguisher $D$ has $\mathsf{Adv}_D(X_0, X_n) = \epsilon$, then for some $1 \leq i < n$ we have that $\mathsf{Adv}_D(X_i, X_{i+1}) \geq \epsilon/n$.*

**Proof:** The proof is simple but is worth giving. We give it for the last quantitive version. Indeed, this implies the fact that $X_0 \approx X_n$, since if $D$ has non-negligible advantage $\epsilon(k)$ on $X_0$ and $X_n$, then $D$ has (still non-negligible as $n$ is polynomial in $k$) advantage $\epsilon(k)/n$ on $X_i$ and $X_{i+1}$, which is a contradiction.

To prove the result, let $p_i = \Pr(D(X_i) = 1)$. Thus, we assumed that $\mathsf{Adv}_D(X_0, X_n) = |p_n - p_0| \geq \epsilon$. But now we can use the following very simply algebra:

$$\begin{aligned}
\epsilon &\leq |p_n - p_0| \\
&= |(p_n - p_{n-1}) + (p_{n-1} - p_{n-2}) + \ldots + (p_2 - p_1) + (p_1 - p_0)| \\
&\leq |p_n - p_{n-1}| + |p_{n-1} - p_{n-2}| + \ldots + |p_2 - p_1| + |p_1 - p_0| \\
&= \sum_{i=0}^{n-1} |p_{i+1} - p_i|
\end{aligned}$$

Notice, we simply used algebraic manipulation and nothing else. However, now we see that for some index $i$, we have

$$|p_{i+1} - p_i| \geq \frac{\epsilon}{n}$$

$\square$

Despite its triviality, this lemma is very powerful in the following regard. Assume we wish to prove that $X \approx X'$, but $X$ and $X'$ look somewhat different on the first glance. Assume we can define (notice, its completely our choice!) $X_0 \ldots X_n$, where $n$ is constant or even polynomial in $k$, s.t.

1. $X_0 = X$, $X_n = X'$.
2. For every $1 \leq i < n$, $X_i \approx X_{i+1}$.

Then we conclude that $X \approx X'$. This simple technique is called the *hybrid argument*. The reason some people have difficulty in mastering this simple technique is the following. Usually $X$ and $X'$ are some natural distributions (say $G(x)$ and $R$, as in PRG example). However, the "intermediate" distributions are "unnatural", in a sense that they never come up in definitions and applications. In some sense, one wonders why a distinguisher $D$ between natural $X$ and $X'$ should even work on these "meaningless" distributions $X_i$?

The answer is that $D$ is simply an algorithm, so it expects some input. We are free to generate this input using any crazy experiment that we like. Of course, the behavior of $D$ maybe crazy as well in this case. However, technically it has to produce some binary answer no matter how we generated the input. Of course a really malicious $D$ may try to really do some crazy things if it *can tell* that we did something he does not expect (i.e., feed it $X_i$ instead of $X$ or $X'$ as we were supposed to). But the point is that if for all $i$ we have $X_i \approx X_{i+1}$, $D$ really *cannot tell* that we generated the input according to some meaningless distributions.

As a simple example, we prove the following very useful theorem about PRG's which we call the composition theorem. Now you will see how simple the proof becomes: compare it with a direct proof!

**Theorem 2 (Composition of PRG's)** *If $G_1 : \{0,1\}^k \to \{0,1\}^{p(k)}$ and $G_2 : \{0,1\}^{p(k)} \to \{0,1\}^{q(k)}$ are two PRG's, then their composition $G : \{0,1\}^k \to \{0,1\}^{q(k)}$, defined as $G(x) = G_2(G_1(x))$, is also a PRG.*

**Proof:** We know that $G_1(x) \approx r$ and $G_2(r) \approx R$, where $x \in \{0,1\}^k$, $r \in \{0,1\}^{p(k)}$ and $R \in \{0,1\}^{q(k)}$ are all random in their domains. We have to show that $G(x) = G_2(G_1(x)) \approx R$. We use a hybrid argument and define an intermediate distribution $G_2(r)$. First, since $G_2$ is polynomial time and $G_1(x) \approx r$ (as $G_1$ is a PRG), then by Lemma 2 we have $G_2(G_1(x)) \approx G_2(r)$. Combining with $G_2(r) \approx R$ (as $G_2$ is a PRG), we use Lemma 3 (i.e., the hybrid argument) to conclude that $G_2(G_1(x)) \approx R$, i.e. that $G$ a PRG. □

Finally, so far we said that $X_0 \approx X_1$ if no distingusiher $D$ can "behave noticeably differently" when given a sample of $X_0$ as opposed to a sample of $X_1$. Here is anollowing equivalent view of this fact, stating that $D$ can behave differently on $X_0$ and $X_1$ only if it effectively can tell whether or not it is given a sample of $X_0$ as opposed to sample of $X_1$ with probability noticeably different from 1/2.

**Lemma 4** *$X_0 \approx X_1$ if and only if, for any efficient distingusisher $D$,*

$$\left| \Pr(D(Z) = b \mid b \xleftarrow{r} \{0,1\}, Z \xleftarrow{r} X_b) - \frac{1}{2} \right| \leq \mathsf{negl}(k)$$

**Proof:** We have

$$\begin{aligned} \left| \Pr(D(Z) = b \mid b \xleftarrow{r} \{0,1\}, Z \xleftarrow{r} X_b) - \frac{1}{2} \right| &= \frac{1}{2} \cdot |\Pr(D(X_1) = 1) + \Pr(D(X_0) = 0) - 1| \\ &= \frac{1}{2} \cdot |\Pr(D(X_1) = 1) - \Pr(D(X_0) = 1)| \end{aligned}$$

□

In the sequel we will interchangeably use both of these equivalent formulations of indistinguishability.

# 3 Next-Bit ⇒ PRG (Proof of Theorem 1)

Before proving this result, let us introduce some useful notation. Assume the input $x$ is chosen at random from $\{0,1\}^k$. Let $n = p(k)$, $G(x) = g_1 \dots g_n$ be the output of $G$, and $G_i = g_1 \dots g_i$ be the first $i$ bits of $G(x)$. We also denote by $R_s$ a truly random string of length $s$. We will also often omit the concatenation sign (i.e., write $G_i R_{n-i}$ in place of $G_i \circ R_{n-i}$).

We will split the proof into two steps. The first step uses the hybrid argument to reduce our problem to showing indistinguishability of $n$ pairs of distributions, each related to some specific output bit $1 \le i \le n$ of $G$. Later we will show that each of these pairs is indeed indistinguishable by using the unpredictability of the corresponding bit $i$ of $G$.

## 3.1 Stage 1: Hybrid Argument

Let us see what we have to show. We have to show that $G(x) \approx R$, which in our notation means $G_n \approx R_n$. We use the hybrid argument with the following intermediate distributions:

$$X_0 = R_n,\ X_1 = G_1 R_{n-1}\ , \dots,\ X_i = G_i R_{n-i}\ , \dots,\ X_{n-1} = G_{n-1} R_1\ ,\ X_n = G_n$$

More graphically,

$$\begin{array}{rcccccccccc}
R_n & = & \boxed{r_1} & r_2 & \dots & r_{i-1} & r_i & r_{i+1} & \dots & r_{n-1} & r_n \\
G_1 R_{n-1} & = & \boxed{g_1} & r_2 & \dots & r_{i-1} & r_i & r_{i+1} & \dots & r_{n-1} & r_n \\
\vdots & & & & \vdots & & & & \vdots & & \\
G_{i-1} R_{n-i+1} & = & g_1 & g_2 & \dots & g_{i-1} & \boxed{r_i} & r_{i+1} & \dots & r_{n-1} & r_n \\
G_i R_{n-i} & = & g_1 & g_2 & \dots & g_{i-1} & \boxed{g_i} & r_{i+1} & \dots & r_{n-1} & r_n \\
\vdots & & & & \vdots & & & & \vdots & & \\
G_{n-1} R_1 & = & g_1 & g_2 & \dots & g_{i-1} & g_i & g_{i+1} & \dots & g_{n-1} & \boxed{r_n} \\
G_n & = & g_1 & g_2 & \dots & g_{i-1} & g_i & g_{i+1} & \dots & g_{n-1} & \boxed{g_n}
\end{array}$$

Since $n = p(k)$ is polynomial in $k$ and $X_0 = R_n$, $X_n = G_n$, by the hybrid argument we only have to show that for every $1 \le i \le n$, we have $G_{i-1} R_{n-i+1} \approx G_i R_{n-i}$. We will do it in the next step, but notice (see the table above) how similar these two distributions are: they are only different in the $i$-st bit. In other words, both of them are of the form $G_{i-1} b R_{n-i}$, where $b$ is either the "next bit" $g_i$ or a truly random bit $r_i$. Not surprisingly, the fact that they are indistinguishable comes from the unpredictability of $g_i$ given $G_{i-1}$. Looking ahead, $G_{i-1}$ is the legal input to our next bit predictor $P$, while $R_{n-i}$ can be easily sampled by the predictor $P$ itself!

### 3.2 Stage 1.5: defining the predictor

To complete the proof, we have to show that $G_{i-1}R_{n-i+1} \approx G_iR_{n-i}$. For this it suffices to show that if there exists a PPT distinguisher $A$ for the above two distributions (that has non-negligible advantage $\delta$), then there exists a PPT next-bit predictor $P$ for $g_i$ given $G_{i-1}$, which would contradict the next-bit unpredictability for $G$. So assume such $A$ exists. Assume without loss of generality that $\Pr[A(G_{i-1}R_{n-i+1}) = 0] = q$, and that $\Pr[A(G_iR_{n-i}) = 0] = q + \delta$ (that is, we are assuming w.l.o.g. that $A$ outputs 0 more often when the $i$-th bit is from $G$ rather than random; if not, simply rename $q$ to $1-q$ and swap 0 and 1 in the output of $A$). Now, some shortcuts in notation.

Whenever we will run $A$, the first $(i-1)$ bits come from the generator, last $(n-i)$ bits are totally random, i.e. only the $i$-th bit is different. So we denote by $A(b)$, $b \in \{0,1\}$, running $A(G_{i-1}bR_{n-i})$. Note, that when running $A(b)$ several times, we always leave the *same prefix* $G_{i-1}$ that was given to us at the beginning, but always put brand new random bits in the last $(n-i)$ positions. Now we denote by $r \in \{0,1\}$ a random bit (to represent $r_i$) and by $g = g_i$ — the $i$-th bit of $G$, where the seed is chosen at random. Hence, we know that

$$\Pr(A(g) = 0) - \Pr(A(r) = 0) \geq \delta$$

Now, let us recap where we stand. We are trying to build $P$ that will guess $g$. $P$ can run $A(0)$ or $A(1)$. $P$ knows that $A(g)$ is more likely to be 0 than $A(r)$ (for a random bit $r$). So how can $P$ predict $g$? It turns out that there are several ways that work. Here is one of them.

$P$ picks a random $r$ and runs $A(r)$. If the answer is 0, it seems likely that $g = r$, since $A(g)$ is more likely to be 0 than $A(r)$. So in this case $P$ guesses that $g = r$ (i.e. outputs the value of $r$). If, on the other hand, $A(r)$ returns 1, it seems like it is more likely that $g$ is the compliment of $r$, so we guess $g = 1 - r$. This is our entire predictor, and let us call its output bit $B$. We wish to show that $\Pr[B = g] \geq \frac{1}{2} + \delta$.

To put our intuition differently, $A(r)$ *a-priori* outputs 0 less often than $A(g)$. Thus, if $A(r)$ returned 0, this gives us some *a-posteriori* indication that $r = g$.

### 3.3 Stage 2: proving our predictor is good

Let us now show that $P$ works. The proof is quite technical. Keep in mind though, that what we are doing is simply an exercise in probability, our intuition is already in place!

Let $z = \Pr[g = 0]$ (where the probability is over random seed $x$). We introduce the following "irreducible" probabilities:

$$\beta_{jk} := \Pr[A(j) = 0 \mid g = k], \quad j,k \in \{0,1\} \tag{3}$$

The reason that this probabilities are important is that we will have to analyze the expression $\Pr[P(G_{i-1}) = g]$, and therefore, will have to immediately condition on the value of $g$, i.e. $g = 0$ or $g = 1$. And since $P$ runs $A$, the needed probability will indeed be some function of $z$ and $\beta_{jk}$'s. We note that all 4 probabilities in (3) are generally different. Indeed, conditioning on a particular setting of $g$ skews the distribution of the first $(i-1)$

bits $G_{i-1}$. We start by expressing our given probabilities in terms of "irreducible" ones (in both formulas in the last step we condition over $g = 0$ or $g = 1$):

$$\begin{aligned} q &= \Pr[A(r) = 0] \\ &= \frac{1}{2}(\Pr[A(0) = 0] + \Pr[A(1) = 0]) \\ &\overset{(3)}{=} \frac{1}{2}(z\beta_{00} + (1-z)\beta_{01} + z\beta_{10} + (1-z)\beta_{11}) \\ q+\delta &= \Pr[A(g) = 0] \\ &\overset{(3)}{=} z\beta_{00} + (1-z)\beta_{11} \end{aligned}$$

Subtracting the first equation from the second, we get the main equality that we will use:

$$\delta = \frac{z(\beta_{00} - \beta_{10}) + (1-z)(\beta_{11} - \beta_{01})}{2} \tag{4}$$

Now, let us return to the analysis of $P$ (recall, it chooses a random $r$, runs $A(r)$ and then decides if it output $B = r$ or $B = 1 - r$ depending on whether or not the answer is 0). The probabilities of $P$'s success for a *fixed* $r = 0$ or $r = 1$ are:

$$\begin{aligned} \Pr[B = g \mid r = 0] &= z\Pr[A(0) = 0 \mid g = 0] + (1-z)\Pr[A(0) = 1 \mid g = 1] \\ &\overset{(3)}{=} z\beta_{00} + (1-z)(1-\beta_{01}) \\ &= (1-z) + z\beta_{00} - (1-z)\beta_{01}. \\ \Pr[B = g \mid r = 1] &= z\Pr[A(1) = 1 \mid g = 0] + (1-z)\Pr[A(1) = 0 \mid g = 1] \\ &\overset{(3)}{=} z(1-\beta_{10}) + (1-z)\beta_{11} \\ &= z - z\beta_{10} + (1-z)\beta_{11}. \end{aligned}$$

Hence, conditioning on random bit $r$, the overall probability of $P$'s success is

$$\begin{aligned} \Pr[B = g] &= \frac{1}{2}\Pr[B = g \mid r = 0] + \frac{1}{2}\Pr[B = g \mid r = 1] \\ &= \frac{1}{2}\left(z + (1-z) + z\beta_{00} - (1-z)\beta_{01} - z\beta_{10} + (1-z)\beta_{11}\right) \\ &= \frac{1}{2} + \frac{z(\beta_{00} - \beta_{10}) + (1-z)(\beta_{11} - \beta_{01})}{2} \\ &\overset{(4)}{=} \frac{1}{2} + \delta \end{aligned}$$

This completes the proof. One remark is in place, though. Despite its technicality, the proof is quite intuitive. Unfortunately, it seems like the "right" expressions are magically appearing at the "right" places. This is just an illusion. There are several other intuitive predictors, and all come up to the same expressions. Unfortunately, having four probabilities $\beta_{jk}$ indeed seems to be necessary.

## 4 Consequences

Having proven that next-bit unpredictability $\Rightarrow$ PRG, and using Lemma 1, we get

**Corollary 5** *$G'$ and $G$ defined by Equation (1) and Equation (2) are* PRG*'s.*

Notice, however, that $G'$ and $G$ are very inconvenient to evaluate. Specifically, we (1) have to know $n$ in advance, and (2) have to output the bits in reverse order, so that we have to wait for $n$ steps before outputting the first bit. It would be much nicer if we could output the hardcore bits in the natural "forward order". But now, using Corollary 5, we can! Indeed, redefine

$$G'(x) = h(x) \circ h(f(x)) \circ \ldots \circ h(f^{n-1}(x)) \quad (5)$$

$$G(x) = G'(x) \circ f^n(x) = h(x) \circ h(f(x)) \circ \ldots \circ h(f^{n-1}(x)) \circ f^n(x) \quad (6)$$

From the definition of a PRG, it is clear that the order of the output bits does not matter — a PRG remains pseudorandom no matter which order we output its bits. For the sake of exercise, let us show this formally for $G'$ (similar argument obviously holds for $G$). Clearly, it suffices to show that

**Lemma 6** *If $F(x) = g_1 \ldots g_n$ is a* PRG*, then so is $H(x) = g_n \ldots g_1$.*

**Proof:** Let $rev(g_1 \ldots g_n) = g_n \ldots g_1$. Since $rev$ is poly-time computable, by Lemma 2 we have $H(x) \equiv rev(G(x)) \approx rev(R) \equiv R$, showing that $H$ is a PRG. □

**Theorem 3** *$G'$ and $G$ defined by Equation (5) and Equation (6) are* PRG*'s, provided $f$ is a* OWP *and $h$ is its hardcore bit.*

Notice, we can now evaluate $G'$ and $G$ much more efficiently. Simply keep state $s = f^i(x)$ after outputting $i$ bits, then output bit $h(s)$ as the next bit, and update the state to $s = f(s)$. Moreover, to evaluate $G'$ we do not even need to know $n$ in advance! We can get as many bits out as we wish! On the other hand, the moment we are sure we do not need many more bits from $G'$, we can simply output our current state $s = f^n(x)$ (for some $n$), and get the output of $G$ instead. This will save us $k$ evaluations of our OWP. However, it seems like using $G'$ is still much better that using $G$ since we can keep going forever (with $G$, we cannot go on with the current seed $x$ the moment we reveal $f^n(x)$). The pseudo-code is summarized below.

Pick a random seed $x_1 \leftarrow^r \{0,1\}^k$;
**repeat until no more bits are needed**
  output next bit pseudorandom bit $b_i = h(x_i)$;
  update the seed $x_{i+1} := f(x_i)$;
  $i := i + 1$;
**end repeat**
If want the last chunk of $k$ bits, output the current seed $x_i$;

## 5 Examples

We discuss two specific examples of pseudorandom generators induced from familiar OWP's.

### 5.1 Blum/Micali Pseudo-Random Generator

The Blum/Micali generator uses the candidate OWP $EXP_{p,g}(x)$ to generate a pseudo-random string. Namely, we recognize that if $x_{i+1} = g^{x_i} \bmod p$, the $MSB(x)$ is always hardcore.

### 5.2 Blum/Blum/Shub Pseudo-Random Generator

Next, we look at the Blum/Blum/Shub Generator, which uses the proposed OWF $SQ_n(x) = x^2 \bmod n$ where $n$ is a Blum integer (ie the product of two primes $p$ and $q$ such that $p \equiv q \equiv 3 \bmod 4$. The restriction on $n$ to be a Blum integer comes from the fact that $(x^2 \bmod n)$ becomes a *permutation* when restricted to the subgroup of quadratic residues $QR_n$ of $\mathbb{Z}_n^*$. We mentioned that under the assumption that $SQ_n : QR_n \to QR_n$ is a OWP, the $LSB(x)$ is a hardcore bit for $SQ_n$, and this defined the Blum-Blum-Shub generator. Notice, $x_{i+1} = x^{2^i} \bmod n$. As you show in the homework, $x_{i+1}$ is very easy to compute directly when given the factorization of $n$ (i.e. without iterating $SQ_n$ for $i$ times). Also, each next bit requires only one modular multiplication. Finally, one can show that it is safe to use simultaneously even up to $\log k$ least significant bits of each $x_i$, making the BBS generator even more efficient.

By comparison we look at a linear congruential generator of the form based on $(f_n(x) = (ax + b) \bmod n)$, where $a, b$ are chosen at random, which seems very similar but which in fact proves insecure. The contrast shows us the importance of building a general theory. We see that BBS is secure since $LSB(x)$ is a hardcore bit for $SQ_n$, while one of the reasons the once-popular linear congruential generator is insecure, is the fact that $LSB(x)$ is not its hardcore bit. Without this understanding, we would have hard time a-priori to say which generator is better.

## 6 PRG's and OWF's

First, we notice that

**Lemma 7** *The existence of a* PRG *which stretches* $\{0,1\}^k \to \{0,1\}^{k+1}$ *implies the existence of a* PRG *which stretches* $\{0,1\}^k \to \{0,1\}^{p(k)}$

For example, this follows from the repeated iteration of the composition theorem Theorem 2: simply call $G$ repeatedly on its own output for $p(k)$ times (notice, the adversary's advantage increases by a polynomial factor $p(k)$, and hence remains negligible). Thus we see that the *assumption* that PRG exists is *universal* and the *actual expansion is unimportant.*

But now we can ask the question of finding the necessary and sufficient conditions for the existence of PRG's. We proved that OWP $\Rightarrow$ PRG. In your homework you show that PRG $\Rightarrow$ OWF. There is a very celebrated result (omitted due to its extremely complicated proof) which shows that in fact OWF $\Rightarrow$ PRG. This gives us the dramatic conclusion that

**Theorem 4** OWF $\iff$ PRG

This is extremely interesting because we see that two of the main primitives that we have introduced (namely OWF and PRG) are in fact equivalent despite their disparate appearances.

## 7 Forward Security

Next we introduce the important notion of *forward security*. First, recall the iterative construction of $G'(x)$ in Equation (5). Every time we output the next bit $b_i = h(x_i)$, we also update our state to $x_{i+1} = f(x_i)$. We also noticed that at every moment of time $n$, we can simply output $x_{n+1} = f^n(x)$ and still get a secure PRG $G(x)$. To put it in different context, our current state $x_{n+1}$ looks completely uncorrelated with the previously output bits $G'(x)$. Namely, *even if we manage to loose or expose our current state, all the previously output bits remain pseudorandom*!

This is exactly the idea of forward-secure PRG's formally defined below.

Definition 4 [Forward Secure PRG] A forward-secure PRG with block length $t(k)$ is a poly-time computable function $F : \{0,1\}^k \to \{0,1\}^k \times \{0,1\}^{t(k)}$, which on input $s_i$ — the current state at period $i \geq 1$ — outputs a pair $(s_{i+1}, b_i)$, where $s_{i+1}$ is the next state, and $b_i$ are the next $t(k)$ pseudorandom bits. We denote by $N(s)$ the next-state function, by $B(s)$ the next $t(k)$ pseudorandom bits output, by $F_i(s_1) = B(s_1)B(s_2)\dots B(s_i)$ the pseudorandom bits output so far, and by $R_i$ — a random string of length $\{0,1\}^{i \cdot t(k)}$. We require for any $i < \mathsf{poly}(k)$ that when the initial state $s_1$ is chosen at random from $\{0,1\}^k$, we have

$$(F_i(s_1), s_{i+1}) \approx (R_i, s_{i+1})$$

♢

For example, when used for symmetric key encryption, a forward-secure generator implies that loosing the current key leaves all the previous "one-time pad" encryptions secure.

We notice that our generic PRG $G'$ from Equation (5) and its efficient implementation naturally leads to a forward-secure generator with block length 1: $F(s) = (f(s), h(s))$, i.e. the next state is $f(s)$ and the next bit is $h(s)$. The proof of forward security immediately follows from the fact that $G$ from Equation (6) is a PRG (check this formally as an exercise).

Finally, we notice that one can also build a forward-secure PRG with any polynomial block length $t(k)$ from any regular PRG $G : \{0,1\}^k \to \{0,1\}^{t(k)+k}$. If we let $G(s) = G_1(s) \circ G_2(s)$, where $|G_1(s)| = |s| = k$, then $G$ by itself is a forward-secure generator $G(s) = (G_1(s), G_2(s))$. Namely, we use $G_1(s)$ as the next state, and $G_2(s)$ as the $t(k)$ bits we output.

**Theorem 5** *Forward-secure* PRG*'s with any polynomial block length* $1 \leq t(k) < \mathsf{poly}(k)$ *exist* $\iff$ *regular* PRG*'s exist (*$\iff$ OWF*'s exist).*

We leave the proof of this fact as an exercise.

## 8 PRG's in Our Lives

As a final note we would like to emphasize how common (and important!) PRG's are in real life. In computing, most requests for "random" sequence in fact access a *pseudorandom* sequence. Indeed, to uise randomness in most computer languages we first make a call to the function randomize, which initializes some PRG using only a small (and hopefully) truly random seed. All the subsequent random calls are in fact deterministic uses of a PRG, which

outputs the next "random-looking sequence" (and possibly updates its state). Hence, since what we usually take for random is in fact some deterministic sequence generated by a PRG, we see that it is very important to understand what constitutes a good PRG. Which is exactly what we have spent the past two weeks investigating.

Finally, we recap our discussion of PRG's by reminding that they can be used without much harm in any realistic (i.e., *efficient*) application which expects truly random bits.

| CSCI-GA.3210-001<br>MATH-GA.2170-001 | **Introduction to Cryptography** February 29, 2012 |
|---|---|

Lecture 6

*Lecturer: Yevgeniy Dodis* *Spring 2012*

# 1 Public-Key Encryption

Last lecture we studied in great detail the notion of pseudorandom generators (PRG), a deterministic functions that stretch randomness by any *polynomial amount*: from $k$ to $p(k)$ bits. As we already indicated, PRG's have a lot of applications including constructions of both public- and private-key encryptions, and implementation of "ideal randomness" in essentially any programming language. In this lecture we will begin examining these application in more detail by starting with the formal study of public-key encryption (PKE). As we explained before, the informal scenario is this:

- **Before the Encryption.** Alice publishes to the world her public key $PK$. Therefore, both Bob and Eve know what $PK$ is. This public key is only used to encrypt messages, and a separate key $SK$ is used to decrypt messages. (This is unlike the Secret-Key scheme where one key $S$ is used to both encrypt and decrypt.) Only Alice knows what $SK$ is, and nobody else, not even Bob.
- **Encryption.** When Bob wishes to send Alice a plaintext message $M$ via the Internet, Bob encrypts $M$ using Alice's public key $PK$ to form a ciphertext $C$. (Formally, we summarize encryption with $PK$ as $E_{PK}$ and say that $C = E_{PK}(M)$.) Bob then sends $C$ over the Internet to Alice.
- **Decryption.** Upon receiving $C$, Alice uses her secret private key $SK$ to decrypt $C$, giving her $M$, the original plaintext message. (Formally, we summarize decryption with $SK$ as $D_{SK}$ and say that $D_{SK}(C) = D_{SK}(E_{PK}(M)) = M$.)
- **Eve's Standpoint.** Unlike the Secret-Key scheme, Eve knows everything Bob knows and can send the same messages Bob can. And, only Alice can decrypt. And, when Bob sends his message, Eve only sees $C$, and knows $PK$ in advance. But, she has no knowledge of $SK$. And, if it is hard for Eve to learn about $SK$ or plaintexts based on ciphertexts and $PK$, then our system is secure.

# 2 Definition of Public-key Encryption

We start with the syntax of a public-key encryptions scheme, and only later talk about its security. Definition 1 [Public-key encryption (PKE)] A PKE is a triple of PPT algorithms $\mathcal{E} = (Gen, E, D)$ where:

1. *Gen* is the *key-generation algorithm*. $Gen(1^k)$ outputs $(PK, SK, M_k)$, where $SK$ is the secret key, $PK$ is the public-key, and $M_k$ is the message space associated with the $PK/SK$-pair. Here $k$ is an integer usually called the *security parameter*, which determines the security level we are seeking for (i.e., everybody is polynomial in $k$ and adversary's "advantage" should be negligible in $k$).

2. $E$ is the *encryption algorithm*. For any $m \in M_k$, $E$ outputs $c \stackrel{r}{\leftarrow} E(m;\ PK)$ — the encryption of $m$. $c$ is called the *ciphertext*. We sometimes also write $E(m;\ PK)$ as $E_{PK}(m)$, or $E(m;\ r, PK)$ and $E_{PK}(m;\ r)$, when we want to emphasize the randomness $r$ used by $E$.

3. $D$ is the decryption algorithm. $D(c;\ SK) \stackrel{r}{\rightarrow} \tilde{m} \in \{\mathsf{invalid}\} \cup M_k$ is called the decrypted message. We also sometimes denote $D(c;\ SK)$ as $D_{SK}(c)$, and remark that usually $D$ is deterministic.

4. We require the **correctness** property: if everybody behaves as assumed

$$\forall m \in M_k, \quad \tilde{m} = m, \quad \text{that is} \quad D_{SK}(E_{PK}(m)) = m$$

$\diamondsuit$

**Example: RSA.** Let us check that RSA satisfies the above definition. Notice, both $E$ and $D$ are deterministic.

1. $Gen(1^k)$ corresponds to the following algorithm: $(p, q)$ are random primes of $k$ bits, $n = pq,\ e \stackrel{r}{\leftarrow} \mathbb{Z}^*_{\varphi(n)},\ d = e^{-1} \bmod \varphi(n),\ M_k = \mathbb{Z}^*_n$. Set $PK = (n, e),\ SK = d$.

2. $c = E(m;\ (n, e)) = m^e \bmod n$.

3. $\tilde{m} = D(c;\ (d, n)) = c^d \bmod n$.

More generally, we could construct a PKE from any TDP. Suppose we have a TDP $f$ with trap-door information $t_k$ and algorithm $I$ for inversion. Here is the induced PKE:

1. $Gen(1^k) \stackrel{r}{\rightarrow} (f, t_k, \{0,1\}^k)$, and $f$ is the $PK$ and the trapdoor $t_k$ is the $SK$.

2. $E(m;\ PK) = f(m)$.

3. $D(m;\ SK) = I(c, t_k)$.

**Conventions about message spaces $M_k$.** Without loss of generality we will assume that the message space $M_k$ can be determined from the public key $PK$ (so we will not explicitly output its desciption in *Gen*). Also, in many schemes the message space $M_k$ does not depend on the particular public key $PK$ and depends only on $k$, e.g. $M_k = \{0,1\}$ and $M_k = \{0,1\}^k$. In the latter cases we will sometimes say that $\mathcal{E}$ is an encryption for a sequence of message spaces $\{M_k\}$. Notice, however, that in most concrete examples (i.e., in the RSA example above), $M_k$ could depend on the $PK$. As we will see, this will create some definitional issues when defining security of encryption.

**Problems.** The problem of the above general construction is that it does not meet our requirement of security.

- First, it reveals partial information. For example, in the RSA case, $f(m)$ preserves the Jacobi symbol of $m$. Furthermore, if $f'$ is a TDP $f(a,b) = (a, f'(b))$ is also a TDP however it reveals half of the input message.
- Second, our definition of TDP is based on the assumption of uniform distribution of the input. Here, it corresponds to the uniformity of the distribution on the message space. However, in practice, such uniformity is rarely satisfied, and in some interesting cases, the message spaces is actually quite sparse, for example, the English text.
- Third, when the message space is sparse (i.e., sell/buy), this method is completely insecure and can be broken by a simple exhaustive search.
- Many more problems exist. For example, the adversary can tell whether the same message is being sent twice or not.

For completeness, we would like to point out the general construction does satisfy a very weak security notion.

Definition 2 [One-way secure encryption] A PKE $\mathcal{E}$ is called *one-way secure* if it is hard to **completely** decrypt a **random** message. Formally, for any PPT $A$

$$\Pr(A(c) = m \mid (SK, PK) \stackrel{r}{\leftarrow} Gen(1^k), m \stackrel{r}{\leftarrow} M_k, c \stackrel{r}{\leftarrow} E_{PK}(m)) \leq \mathsf{negl}(k)$$

$\diamondsuit$

Here is a simple lemma directly from the definitions of TDP that shows

**Lemma 1** *If $M_k = \{0,1\}^k$ is the domain of a TDP $f$, then the PKE induced from $f$ is one-way secure.*

**Conclusions.**

- Much stronger definition is needed in order not to reveal partial information.
- Encryption scheme cannot be deterministic in order to solve the problem of non-uniform/sparse message space.
- Even starting with 1-bit encryption, $M_k = \{0,1\}$, is interesting and non-trivial.

## 3 Secure encryption of one bit

From the previous section, we discussed the problems with one-way security and the straightforward usage of a TDP. In order to have a stronger definition, let us first begin from from trying to encrypt one bit, i.e. $M_k = \{0,1\}$.

One of the conclusions we had is that the encryption scheme must be probabilistic. For each bit from $M_k = \{0,1\}$, there is cloud of messages in the encrypted message space $C$ corresponding to that bit. Informally, we want the distribution of these two clouds to be

indistinguishable to the adversary *even conditioned on the public key* $PK$ and even though their supports are totally disjoint (the disjointness is from the fact that we want to decrypt the message without ambiguity). We write it as $\langle PK, E_{PK}(0)\rangle \approx \langle PK, E_{PK}(1)\rangle$, where $E_{PK}(0)$ and $E_{PK}(1)$ are two random variables denoting random encryption of 0 and 1 respectively. Notice this is not possible in the Shannon theory, because there the adversary has infinite power and the only way for the distribution to be indistinguishable is that they are exactly the same, which is not the case since the supports of two distributions are totally different. However, in our case it is doable because we assume the adversary is only PPT. We also point out that when the public key $PK$ is clear, we will sometime be sloppy and simply write $E(0) \approx E(1)$, always implicitly assuming that $PK$ is public knowledge. Here is the formal definition.

DEFINITION 3 A PKE for $M_k = \{0,1\}$ is called *polynomially indistinguishable* if $\langle PK, E_{PK}(0)\rangle \approx \langle PK, E_{PK}(1)\rangle$, meaning that $\forall$ PPT $A$

$$\Big|\ \Pr(A(c, PK) = 1 \mid (SK, PK) \xleftarrow{r} Gen(1^k), c \xleftarrow{r} E_{PK}(0)) - \Pr(A(c, PK) = 1 \mid (SK, PK) \xleftarrow{r} Gen(1^k), c \xleftarrow{r} E_{PK}(1))\ \Big| \ \leq \ \mathsf{negl}(k)$$

Or equivalently,

$$\left|\ \Pr(A(c, PK) = b \mid (SK, PK) \xleftarrow{r} Gen(1^k), b \xleftarrow{r} \{0,1\}, c \xleftarrow{r} E_{PK}(b)) - \frac{1}{2}\ \right| \leq \mathsf{negl}(k)$$

$\diamondsuit$

**Example.** Suppose $f$ is a TDP with trapdoor information $t_k$ and efficient algorithm $I$ for inversion, and $h$ is a hardcore bit for $f$. Here is the PKE we informally considered earlier:

1. $Gen(1^k) \xrightarrow{r} (f, t_k)$.
2. $E(b) \to \langle f(x), h(x) \oplus b\rangle = \langle y, d\rangle$. ($x$ is random in $\{0,1\}^k$).
3. $D(\langle y, d\rangle, t_k) :\ x = I(y, t_k), b = d \oplus h(x)$.

Here is another, slightly more efficient suggestion:

1. $Gen(1^k) \xrightarrow{r} (f, t_k)$.
2. $E(b)$: sample $x \xleftarrow{r} \{0,1\}^k$ until $h(x) = b$, then set ciphertext $y = f(x)$.
3. $D(y)$: recover $x \xleftarrow{r} I(y, t_k)$, then decrypt $\tilde{b} = h(x)$.

Notice, $E$ is efficient, since sampling the right $x$ will terminate after approximately two trials, since $h$ must be balanced between 0 and 1. We analyze these schemes formally later (or in the homework).

## 4 Secure Encryption of Many bits

Now, we will consider the case $M_k = \{0,1\}^{p(k)}$, where $p$ is some polynomial in $k$. The definition is an obvious generalization of the "bit" version.

Definition 4 [Polynomial indistinguishability] A PKE $\mathcal{E}$ for $M_k = \{0,1\}^{p(k)}$ is called *polynomially indistinguishable* (against $PK$-only attack) if for any $m_0, m_1 \in M_k$, we have $\langle PK, E_{PK}(m_0)\rangle \approx \langle PK, E_{PK}(m_1)\rangle$. Formally, $\forall$PPT $A$

$$\left| \Pr(A(c, PK) = b \mid (SK, PK) \stackrel{r}{\leftarrow} Gen(1^k), b \stackrel{r}{\leftarrow} \{0,1\}, c \stackrel{r}{\leftarrow} E_{PK}(m_b)) - \frac{1}{2} \right| \leq \mathsf{negl}(k)$$

$\diamondsuit$

**Comments.**

- The definition includes the situation when $m_0$ and $m_1$ are the same. In this case, no matter $b = 0$ or $b = 1$, $E$ and $A$ will know nothing about what $b$ is, because they only see the message $m_b$, which is the same, no matter $b = 0$ or $b = 1$.
- As will will see this definition is extremely robust and prevents a lot of attacks. For example, it also excludes the possibility for the adversary to tell whether a message was being sent twice. Informally, if $A$ could determine this, when given $c$, $A$ can generate $c' \leftarrow E(m_0)$, and see if $c$ and $c'$ correspond to the same message, thus determining if $b = 0$.

**Blum-Goldwasser construction.** In the Blum-Goldwasser construction, as we mentioned earlier, we are given a TDP $f$ with trapdoor $t_k$, inversion algorithm $I$, and a hardcore bit $h$. Recall also that if we let $G(x) = G'(x) \circ f^{(n)}(x)$, where $G'(x) = h(x) \circ h(f^1(x)) \circ \cdots \circ h(f^{(n-1)}(x))$, then both $G$ and $G'$ are PRG's. We define:

1. $PK = f$ and $SK = t_k$.
2. $E(m)$: get $x \stackrel{r}{\leftarrow} \{0,1\}^k$, send $c = (G'(x) \oplus m, f^{(n)}(x))$.
3. $D(c)$: use $t_k$ to get $f^{(n-1)}(x), \ldots, f(x), x$, and use them to calculate $G'(x)$ with hardcore bit function $h$. After we have $G'(x)$, recovering $m$ is clear.

To check the correctness of Blum-Goldwasser construction, we need to prove that for all $m_0$ and $m_1$ (below we omit $PK = f$ since it's fixed)

$$E(m_0) \equiv (f^{(n)}(x), G'(x) \oplus m_0) \approx (f^{(n)}(x), G'(x) \oplus m_1) \equiv E(m_1) \tag{1}$$

In order to prove this, we will instead prove a more general lemma.

**Lemma 2 (One-Time Pad Lemma)** *Let $R$ denote the uniform distribution. Then for all distributions $X, Y$ (not necessarily independent!), if $(X, Y) \approx (X, R)$, then for all $m_0$ and $m_1$ we have $(X, Y \oplus m_0) \approx (X, Y \oplus m_1)$.*

**Proof:** The simplest proof is to notice that for any fixed message $m$, $R \oplus m \equiv R$, where $R$ is a random string. I.e., "random+fixed=random". Thus, since XOR is an efficient operation,

$$(X, Y \oplus m_0) \approx (X, R \oplus m_0) \equiv (X, R) \equiv (X, R \oplus m_1) \approx (X, Y \oplus m_1)$$

□

We see that Lemma 2 indeed implies the needed Equation (1). Indeed, consider $X = f^{(n)}(x)$, $Y = G'(x)$. Then, to apply Lemma 2, we only need to argue that $(f^{(n)}(x), G'(x)) \approx (f^{(n)}(x), R')$, where $R'$ is random. But since $f$ is a permutation and $x$ is random, $f^{(n)}(x)$ is just a random string, so $(f^{(n)}(x), R') \equiv R$, and we just need to show that $G(x) = f^{(n)}(x) \circ G'(x)$ is a PRG, which is precisely what we showed last time. Thus, we get

**Theorem 1** *BG construction above defines a polynomially indistinguishable encryption.*

As a special case, we also get the security of the one-bit version of BG encryption that we considered in the previous section.

**Efficient example: squaring over Blum integers.** Recall, the Blum-Blum-Shub construction of $G'$ uses the OWF $SQ(x) = x^2 \bmod n$. This function is a TDP when $n = pq$ with $p = 3 \bmod 4$ and $q = 3 \bmod 4$. Specifically, it can be proved that it is a permutation on $SQ_n$, and the trapdoor key is the factorization $(p, q)$ of $n = pq$. The associated hardcore bit is the least significant bit of $x$. Now we see that this construction is quite efficient, because in order to encrypt $p(k)$ bits, we only need to do $p(k)$ multiplications mod $n$.

## 5 Key Encapsulation Mechanism

The BG example above follows the following *key encapsulation principle* which we will meet in virtually any public-key encryption scheme. The idea is to derive a "radom-looking" key $s$ and its "encryption", and then use $s$ to "one-time pad the message". For example, in the BG scheme above, the key $s$ was equal to $G'(x)$, while the "encryption" of $s$ was the value $\psi = f^{(n)}(x)$. A bit more formally,

Definition 5 [Key Encapsulation Mechanism (KEM)] A *Key Encapsulation Mechanism* is a triple of PPT algorithms $\mathcal{E} = (Gen, KE, KD)$ where:

1. *Gen* is the *key-generation algorithm*. $Gen(1^k)$ outputs $(PK, SK, M_k)$, where $SK$ is the secret key, $PK$ is the public-key, and $M_k$ is the "key space" associated with the $PK/SK$-pair.

2. $KE$ is the *key encapsulation algorithm*. It takes the public key $PK$ and outputs a pair $\langle \psi, s \rangle \stackrel{r}{\leftarrow} KE(PK)$, where $s \in M_k$ is a key and $\psi$ is called the *ciphertext* representing encryption of $s$. We sometimes write $\langle \psi, s \rangle = KE_{PK}(r)$ to emphasize the randomness $r$ used by $KE$.

3. $KD$ is the key decapsulation algorithm. $KD(\psi;\ SK) \stackrel{r}{\rightarrow} \tilde{s} \in \{\mathsf{invalid}\} \cup M_k$ attempts to extract a key from the ciphertext $\psi$. We sometimes denote $KD(\psi;\ SK)$ as $KD_{SK}(\psi)$, and remark that usually $KD$ is deterministic.

4. We require the **correctness** property: if $\langle \psi, s \rangle \stackrel{r}{\leftarrow} KE(PK)$, then $KD(\psi, SK) = s$.

$\diamondsuit$

As we mentioned, in the BG example, $KE(x)$ uses random $x$ to set $s = G'(x)$, $\psi = f^{(n)}(x)$, while $KD(\psi) = G'(f^{(-n)}(\psi))$.

The usefulness of KEM comes from the fact that it immediately yields a PKE, by using the symmetric key $s$ to encrypt the message $m$. For concreteness, below we assume $M_k = \{0,1\}^n$ for some parameter $n = p(k)$ (like in the BG case), and we will use the one-time pad encryption to encrypt $m$ (although later we will see that any symmetric-key encryption will do!). But first we need a definition of security for KEM.

DEFINITION 6 [Polynomial indistinguishability] A KEM $(Gen, KE, KD)$ for $M_k = \{0,1\}^{p(k)}$ is called *polynomially indistinguishable* (against $PK$-only attack) if for randomly generated $PK$ and $\langle \psi, s \rangle \leftarrow KE(PK)$ we have $\langle PK, \psi, s \rangle \approx \langle PK, \psi, R \rangle$, where $R$ is a fresh random string sampled from $M_k$. Formally, $\forall$PPT $A$

$$\begin{aligned} \Pr(A(\psi, s_b, PK) = b \quad | \quad & (SK, PK) \stackrel{r}{\leftarrow} Gen(1^k), b \stackrel{r}{\leftarrow} \{0,1\}, \\ & (\psi, s_0) \stackrel{r}{\leftarrow} KE(PK), s_1 \stackrel{r}{\leftarrow} \{0,1\}^{p(k)}) \\ & \leq \frac{1}{2} + \mathsf{negl}(k) \end{aligned}$$

$\diamondsuit$

**Lemma 3** *Assume $(Gen, KE, KD)$ is polynomially indistinguishable* KEM *for $\{0,1\}^n$. Then the following* PKE *is polynomially indistinguishable for $\{0,1\}^n$:*

- *key generation $Gen$ is the same as in* KEM.
- *encryption $E(m)$: compute $\langle \psi, s \rangle \leftarrow KE$, and let $c = \langle \psi, m \oplus s \rangle$.*
- *decryption $D(\psi, z) = z \oplus KD(\psi)$.*

**Proof:** The proof immediately follows from the One-Time Pad Lemma, where $X = (PK, \psi)$, and $Y = s$. $\square$

We will see several other applications of the KEM paradigm. In particular, we will see that the one-time pad can be replaced by any (one-time) secure symmetrci-key encryption. As a concrete illustration, we can apply it to the symmetric scheme $E_s(m) = G(s) \oplus m$, where $G$ is any PRG. Although we did not yet prove that this scheme is secure, we directly prove that it yields a good KEM.

**Lemma 4** *Assume $(Gen, KE, KD)$ is polynomially indistinguishable* KEM *for $\{0,1\}^k$ and $G$ is a* PRG *from $\{0,1\}^k$ to $\{0,1\}^n$. Then the following $(Gen', KE', KD')$ is a polynomially indistinguishable* KEM *for $\{0,1\}^n$:*

- *$Gen' = Gen$.*
- *$KE'(PK)$: compute $\langle \psi, s \rangle \leftarrow KE(PK)$, and let $\psi' = \psi$, $s' = G(s)$. Output $\langle \psi', s' \rangle$.*

- $KD'(\psi') = G(KD(\psi'))$.

We leave the proof as a simple exercise in the hybrid argument. We also remark that a trivial generalization of the above result shows that *any* polynomially indistinguishable encryption $(Gen, E, D)$ on $\{0,1\}^k$ can be combined with any secure PRG from $k$ to $n$ bits to directly give a polynomially indistinguishable encryption $(Gen, E', D')$ on $\{0,1\}^n$: simply pick a random $k$-bit key $s$, encrypt it using $E$, and append $G(s) \oplus m$ to obtain the encryption of $m$. This shows that one can, in principle, only show how to encrypt relatively short messages, and then be able to encrypt much longer messages, provided a good PRG is availble.

## 6 General transformation from one bit to many bit

Blum-Goldwasser construction shows that given a TDP, we could transfer many bits, by efficiently generalizing the corresponding original BG scheme to encrypt one bit. More generally, the above discussion following Lemma 4 shows that we only need to encrypt messages roughly as long as the security parameter to be able to encrypt much longer messages. However, suppose we have some PKE scheme for one bit, which possibly does not depend on any TDP and does not seem to obviously generalize to encrypt many bits. In fact, assume that the only thing we know about it is that it is indistinguishable one-bit encryption. Is it possible for us to use this scheme to encrypt many bits without using any other assumptions on this scheme? A naive answer is to regard each bit to be a separate message and encrypt it using the PKE scheme for one bit. Luckily, this naive approach works for *public* key encryption.[1] Formally, let $\mathcal{E} = (Gen, E, D)$ be a polynomial indistinguishable PKE scheme for one bit, we could define a PKE scheme $\mathcal{E}' = (Gen', E', D')$ for $M_k = \{0,1\}^n$ ($n = p(k)$ for some polynomial $p$) as follows:

1. $Gen'(1^k) = Gen(1^k) \to (PK, SK)$, i.e. $PK$ and $SK$ are generated in the same way as before $Gen$, except the message space now is $M_k = \{0,1\}^{p(k)}$. Now, given $m \in M_k = \{0,1\}^n$, we denote $m$ as $m^1 m^2 \cdots m^n$.
2. Define $E'_{PK}(m^1 m^2 \cdots m^n) = (E_{PK}(m^1), E_{PK}(m^2), \cdots, E_{PK}(m^n)) \to c^1 c^2 \cdots c^n = c$.
3. Define $\tilde{m} = D'_{SK}(c^1 c^2 \cdots c^n) = (D_{SK}(c^1), D_{SK}(c^2), \cdots, D_{SK}(c^n))$

And it turns out that this bit-by-bit encryption indeed works!

**Theorem 2** *If $\mathcal{E}$ is polynomially indistinguishable for one bit, then $\mathcal{E}'$ is polynomial indistinguishable for $n = p(k)$ bits.*

**Proof:** Take two messages $m_0$ and $m_1$. We first construct a sequence of intermediate messages that slowly go from $m_0$ to $m_1$:

$$\begin{array}{cccccccc}
M_0 & = & m_0^1 & m_0^2 & \dots & m_0^{n-1} & m_0^n \\
M_1 & = & m_1^1 & m_0^2 & \dots & m_0^{n-1} & m_0^n \\
\vdots & & & & \vdots & & \\
M_{n-1} & = & m_1^1 & m_1^2 & \dots & m_1^{n-1} & m_0^n \\
M_n & = & m_1^1 & m_1^2 & \dots & m_1^{n-1} & m_1^n
\end{array}$$

[1] As we will see, things are a bit more complex in the symmetric key setting.

Notice, $M_0 = m_0$ and $M_n = m_1$. Also, $M_{i-1}$ and $M_i$ differ in at most one bit — bit number $i$. We now define a sequence of distributions

$$C_i \leftarrow E'(M_i) = E(m_1^1) \dots E(m_1^i) E(m_0^{i+1}) \dots E(m_0^n)$$

Using the hybrid argument, in order to prove that (we omit the $PK$ from all the distributions for compactness)

$$E'(m_0) = E'(m_0^1 m_0^2 \cdots m_0^n) \approx E'(m_1^1 m_1^2 \cdots m_1^n) = E'(m_1)$$

i.e. $C_0 \approx C_n$, we only need to show that for any $i$, we have $C_{i-1} = E'(x_{i-1}) \approx E'(x_i) = C_i$. Graphically,

$$\begin{array}{rcccccccc} C_{i-1} & = & E(m_1^1) & \dots & E(m_1^{i-1}) & \boxed{E(m_0^i)} & E(m_0^{i+1}) & \dots & E(m_0^n) \\ C_i & = & E(m_1^1) & \dots & E(m_1^{i-1}) & \boxed{E(m_1^i)} & E(m_0^{i+1}) & \dots & E(m_0^n) \end{array}$$

Now, let $A = E(m_1^1) \circ \dots \circ E(m_1^{i-1})$, $B = E(m_0^{i+1}) \circ \dots \circ E(m_0^n)$. Thus, we only need to show that

$$(PK, E(m_0^i), A, B) \approx (PK, E(m_1^i), A, B)$$

But this is obvious now! Since by our assumption on $(Gen, E, D)$, we have $(PK, E_{PK}(m_0^i)) \approx (PK, E(m_1^i))$, and since both $A$ and $B$ *can be computed in polynomial time with the knowledge of* $PK$ (i.e., $(A, B) = g(PK)$ for some *efficient* function $g$), we immediately get the desired conclusion. □

To recap the whole proof, we used the fact that if one can distinguish between encryptions of two long messages encrypted bit-by-bit, there must be some particular index $i$ that gives the adversary this advantage, but this contradicts the bit security of our base encryption scheme. Also, notice that we loose a polynomial factor $p(k) = n$ in security by using the hybrid argument, but this is OK since $n$ is polynomial.

**Remark 1** *We notice that the above one-bit to many-bit result is false for private-key encryption (which we did not cover formally yet). For example, consider the one-time pad with secret bit $s$ and $E_s(b) = b \oplus s$. We know it is perfectly secure. However, if we are to encrypt two bits using $s$, $c_1 = b_1 \oplus s$ and $c_2 = b_2 \oplus s$, then $c_1 \oplus c_2 = b_1 \oplus b_2$, so we leak information. The key feauture of the public-key encryption that makes the result true is the fact that anyone can encrypt using the public key, which is false in the private-key case (check that this is the place where the proof fails).*

**Remark 2** *Although we stated the result for one-bit to many-bits, it is clearly more general. In particular, if we have an indistinguishable encryption of $b$ bits, then we easily get an indistinguishable encryption of $bn$ bits, for any $n = \mathsf{poly}(k)$, by simply splitting the message into $n$ chucks of $b$ bits each, ach encrypting each chunk separately.*

# 7 Chosen Plaintext Secuirty

We have been careful so far to only state the notion of polynomial indistinguishability for the message space $\{0,1\}^n$. However, most practical encryption scheme have general message spaces, which usually depends on the specific public key we choose at key generation. Initially, it seems like it is trivial to generalize our definition for $\{0,1\}^n$ to general message spaces. However, we will see that the resulting notion is slightly incorrect. In fact, it suffers from both a syntactic criticism and a semantic problem.

Syntactic Criticism. A first issue with this definition of security is that it requires that, for each pair of messages $(m_0, m_1)$, their encryptions are indistinguishable from each other: $\boxed{(\forall m_0, m_1 \in M_k)}$, $\forall$ PPT algorithm $A$

$$\left| \Pr[A(c, PK) = b \mid \boxed{(SK, PK, M_k) \leftarrow_r G(1^k)}, b \leftarrow_r \{0,1\}, c \leftarrow_r E_{PK}(m_b)] - \frac{1}{2} \right| \leq \mathsf{negl}(k)$$

In other words, we are quantifying on the message space $M_k$, even before the random choice of $(SK, PK, M_k)$! In the cases in which the message space $M_k$ is fixed, or just varies as a function of the security parameter $k$ (e.g. when $M_k = \{0,1\}$ or $M_k = \{0,1\}^{p(k)}$), this can be fixed simply "moving $M_k$ out" of the output of the key-generation algorithm $G(1^k)$. And this is exactly what we did. But in other interesting cases, like the RSA and the ElGamal PKEs we study later, the message space $M_k$ is indeed somehow related to the public key being used, so that it doesn't yet make sense to choose a pair of messages before knowing the actual public key $PK$ being selected.

This is clearly just a syntactic problem, so that it doesn't affect too much the overall correctness of our current approach. Still it needs to be fixed.

Semantic Criticism. A semantic problem with the notion of indistinguishable security is that it just states that any PPT adversary must have a negligible advantage in distinguishing the encryptions of any pair of messages $(m_0, m_1)$, when the public key is *randomly* selected. Therefore, it does not cast away the chance that, given knowledge of the public key, an adversary may be able to come up with a pair of messages such that it can indeed distinguish the associated ciphertexts.

To overcome this problem, we could be tempted to modify the definition in such a way that first we fix the public key $PK$ and the private key $SK$, and then we quantify overall the possible pair of messages. But in this case we are asking too much, since the existence of "bad" pairs of messages is unavoidable. To see why, consider an adversary that, in distinguishing between the encryptions of $(m_0, m_1)$, always assumes that the second message is the private key $SK$, and tries to decrypt the encryption it has being given accordingly. Clearly, the advantage of this adversary is negligible whenever the second message is not the secret key; however, in the very specific, pathological case in which the pair is of the form $(m_0, SK)$, its advantage will be 1.

After all, what we really want from our definition of security is that, even if the adversary already knows $PK$, it should be infeasible for him to find two messages for which it has a non-negligible advantage.

Security Against Chosen Plaintext Attack. Both the criticisms stated above stem from the the fact that the messages $m_0, m_1$ are quantified "outside" the probability which constitutes the advantage of the adversary:

- from a syntactic point of view, this is bad since the message space $M_k$ is chosen "inside" the probability;
- from a semantic point of view, this is bad since we want to allow the adversary to choose the pair of messages after seeing the $PK$, and thus the probability should be taken over all the pairs of messages that it can efficiently find.

We can fix both this problems with the following definition of:

Definition 7 [Indistinguishable Security Against Chosen Plaintext Attack]
A PKE $\mathcal{E} = (G, E, D)$ is *indistinguishable against a chosen plaintext attack (*CPA*)*, or, shortly, IND-CPA-*secure*, if $\forall$ PPT algorithm $A$

$$\left| \Pr\left[ b = \tilde{b} \;\middle|\; \begin{array}{c} (PK, SK, M_k) \leftarrow_r G(1^k), (m_0, m_1, \alpha) \leftarrow A(PK, M_k, \text{'find'}) \\ b \leftarrow_r \{0,1\}, c \leftarrow E_{PK}(m_b), \tilde{b} \leftarrow A(c, \alpha, \text{'guess'}) \end{array} \right] - \frac{1}{2} \right| = \mathsf{negl}(k)$$

$\diamondsuit$

Let's take a closer look at what is going on! This notion of security can be viewed as the following game between us and the adversary:

1. we run the key-generation algorithm, obtaining $(PK, SK, M_k)$;
2. we give the public key $PK$ and the message space $M_k$ to the adversary and ask it to 'find' a pair of messages in $M_k$ for which it believes it can distinguish encryptions under the key $PK$;
3. the adversary outputs a pair of messages $(m_0, m_1)$ of its choice, along with its final "state", i.e. some kind of summary of the considerations and computations that led it to choose this specific pair of messages;
4. we choose which one of two message we want encrypt, and give it the corresponding ciphertext $c$;
5. we ask the adversary to tell which message we encrypted, allowing it to "remember" the reasons for which it chosen the pair $(m_0, m_1)$.

We win the game (i.e. the considered PKE is IND secure against CPA) if the adversary's advantage (defined as usual as its probability of success) is negligible.

As we will see, the resulting notion is really the "right" notion of security for PKE. Moreover, it turns out it is better suited even for the case when the message space is $\{0,1\}^n$. Namely, we show that this new notion of security is strictly more general than the PK-only security, even when restricted to $\{0,1\}^n$ for which PK-only security also makes sense.

**Theorem 3** (CPA $\Rightarrow$ PK-only)
*If a* PKE $\mathcal{E} = (G, E, D)$ *is* CPA *secure, then* $\mathcal{E}$ *is also* PK*-only secure.*

**Proof:** Let us assume that $\mathcal{E}$ in not PK-only secure, i.e.
$\exists m_0, m_1$, $\exists$ a PPT algorithm $A$ such that

$$\Pr(A(c, PK) = 1 \mid (SK, PK) \leftarrow_r G(1^k), c \leftarrow_r E_{PK}(c)) = \epsilon$$

where $\epsilon$ is not-negligible.

Now, we can construct an adversary $A'$ that, using $A$ as a black-box module, breaks the CPA security of $\mathcal{E}$. Recall that, in the CPA notion of security, the adversary acts in two rounds:

**find** Regardless of the public key $PK$ at hand, $A'$ always chooses the pair of message $(m_0, m_1)$ of the hypothesis, and records in $\alpha$ the public key $PK$, $m_0$ and $m_1$:

$$A'(PK, M_k, \text{`find'}) \rightarrow (m_0, m_1, \alpha)$$

**guess** When challenged to determine which message was encrypted, $A'$ runs $A(c, PK)$:

$$A'(c, \alpha, \text{`guess'}) \rightarrow A(c, PK)$$

Clearly, this algorithm $A'$ has the same (non-negligible) advantage as $A$, contradicting the hypothesis that $\mathcal{E}$ was CPA secure. □

**Theorem 4 (PK-only $\not\Rightarrow$ CPA)**
*There exists a* PKE $\mathcal{E} = (G, E, D)$ *which is* PK*-only secure, but is not* CPA *secure.*

**Proof:** To prove the theorem it suffices to come up with a counterexample, namely a specific PKE $\mathcal{E}'$ which is PK-only secure but not CPA secure. To this purpose, consider an arbitrary PKE $\mathcal{E} = (G, E, D)$ which is PK-only secure and modify it as follow:

**Key-Generation Algorithm**

$$\begin{aligned} &\textbf{set:} && (PK, SK) = G(1^k), r \text{ random} \\ &\textbf{output:} && (PK', SK') = ((PK, r), SK) \end{aligned}$$

**Encryption**

$$E'(m) = \begin{cases} (0, E(PK')), & \text{if } m = PK' \\ (1, E(m)), & \text{otherwise} \end{cases}$$

**Decryption**

$$\begin{cases} D'(0, y) = PK' \\ D'(1, y) = D(y) \end{cases}$$

Clearly, $\mathcal{E}'$ is *not* CPA secure, since once the adversary knows the public key $PK'$, it can ask to be challenged on the pair of messages $(PK', m_1)$, and it will always succeed in distinguishing an encryption of $PK'$ (which starts with 0) from an encryption of any other message (which starts with 1).

Nevertheless, $\mathcal{E}'$ is still a PK-only secure PKE, since, in this setting, it is no longer possible to efficiently find a pair of messages $(m_0, m_1)$ whose encryptions are easy-to-distinguish. Indeed, due to the random $r$ present in the public key $PK'$, the probability that the adversary could guess the public key $PK'$ that will be used, is always negligible (no matter how short the public key $PK$ for the PKE $\mathcal{E}$ could be). Besides, it will be difficult for the adversary to find a generic pair $(m_0, m_1)$ which makes its task easy, since such a pair could be also used to tell apart encryptions for the PKE $\mathcal{E}$, contradicting the hypothesis that $\mathcal{E}$ is PK-only secure. □

**Remark 3** *Despite the slight differences, and the non-equivalence proved in the above theorem, the two notions of security formalized are quite close. In particular, all the results stated so far (the generalization of a one-bit secure* PKE *to a secure* PKE *for any number of bits, the Blum-Goldwasser system, the analysis of security of the ElGamal cryptosystem) hold as well when we consider the indistinguishable security against* CPA*.*

*This is because the separation between* CPA *and* PK*-only security is somewhat artificial, and not of real concern, but anyway, once you know, why not to use the right one?*

## 8 Semantic Security

We defined the notion of indistinguishability for PKE. For convenience, we repeat it here in a slightly modified (but equivalent form), where we "split" the attacker $B$ into $B_1$ (corresponding to $B$ in the 'find' stage) and $B_2$ (corresponding to $B$ in the 'guess' stage)"

Definition 8 [Indistinguishability] A cryptosystem $(G, E, D)$ is called IND-*secure* (against CPA attack) if for any PPT adversary $B = (B_1, B_2)$, we have

$$\Pr[\, b = \tilde{b} \;\mid\; \begin{array}{l}(PK, SK) \leftarrow G(1^k);\\ (m_0, m_1, \beta) \leftarrow B_1(PK);\\ b \leftarrow \{0,1\};\\ \tilde{c} \leftarrow E(m_b;\ PK);\\ \tilde{b} \leftarrow B_2(\tilde{c}, \beta, PK)\ ]\end{array} \qquad \leq \tfrac{1}{2} + \mathsf{negl}(k)$$

♢

The notion of indistingusihability is pretty simple, but it is not clear if it is enough for all applications of encryption. Why are there two messages only, and why is the message chosen to be one of them with probability precisely 1/2? And does it imply "security" when message is chosen from a higher entropy distribution? Motivated by these questions, we now define a seemingly much more powerful definition of *semantic security*. It is more difficult to grasp, but it literally says that the knowledge of the ciphertext cannot help the adversary.

Definition 9 [Semantic security] A cryptosystem $(G, E, D)$ is *semantically secure* (against CPA attack) if for any PPT environment Env and any PPT adversary $A$ there exists a PPT simulator $S$ such that $|p_a(k) - p_s(k)| = \mathsf{negl}(k)$, where

$$
\begin{aligned}
p_a(k) \quad = \quad & \Pr[\, R(m,z)=1 \;\mid\; (PK,SK) \leftarrow G(1^k); \\
& \qquad\qquad\quad\; (m,R,\alpha) \leftarrow \mathsf{Env}(PK) \\
& \qquad\qquad\quad\; c \leftarrow E(m;\ PK); \\
& \qquad\qquad\quad\; z \leftarrow A(c,\alpha,PK)\ ]; \\
\\
p_s(k) \quad = \quad & \Pr[\, R(m,z)=1 \;\mid\; (PK,SK) \leftarrow G(1^k); \\
& \qquad\qquad\quad\; (m,R,\alpha) \leftarrow \mathsf{Env}(PK) \\
& \qquad\qquad\quad\; z \leftarrow S(\alpha,PK)\ ].
\end{aligned}
$$

$\diamondsuit$

Let us trace the meaning behind this definition. First, we explain the constructs used in the definition.

Env stands for the environment/context in which the sender, the receiver, and the adversary operate. In the environment, an event may occur that will cause the sender to need to send a certain message to the receiver. E.g. changes in the enemy's strategies may create the need of two army divisions to communicate. When such an event occurs, the environment will generate the message $m$ the sender will have to send and some partial information $\alpha$ the adversary will infer from knowing the event. Notice, such environment could be arbitrary (albeit PPT), so we quantify over all such environments.

The adversary $A$, as usual, attempts to gather some (more) information about the message from the ciphertext and the partial information $\alpha$. This information is referred to as $z \in Z$. The adversary is considered successful if his algorithm $A$ outputs information $z$ that is relevant, i.e. it has to do with the message $m$. In other words, the message $m$ and the adversary's output $z$ must be related. Formally this means that there is a binary relation $R \subset M_k \times Z$ such $A$ tries to satisfy $R(m,z)$. The environment produces this relation $R$ in addition to the message and information $\alpha$ (since Env is arbitrary, this gives a lot of freedom in generating $R$ that the adversary happens to be "interested in").

In the real world (as opposed to the simulated experiment described below), the sender encrypts $c \leftarrow E_{PK}(m)$, and the adversary intercepts $c$. The adversary has access to $c$, $PK$, and the public information $\alpha$ he learns from the environment. The adversary runs his algorithm that outputs $z$, the information that the adversary hopes would reveal something from the relevant to the message $m$. The probability of the adversary's success is $p_a = \Pr[R(m,z)=1]$, i.e. the probability that the output information $z$ is in relation $R$ with the message $m$.

Now let us discuss the simulated experiment. A simulator $S$ is an algorithm very similar to the one the adversary uses ($A$): $S$'s objective and output is the information $z$ as related to the original message as possible. The only difference is that *$S$ does not receive $c$ as its input*. The simulator operates purely on the public knowledge $\alpha$. Given a paraticular simulator $S$, its probability of success is $p_s(k) = \Pr(R(m,z)=1)$.

Now, what we want to say is that no PPT adversary $A$ can benefit too much from intercepting the ciphertext $c$. Namely, for any such adversary $A$ where exists a simulator $S$, whcih achieves essentially the same probability of success, without any knowledge of $c$! In other words, the knowledge of $c$ is useless for all practical purposes: *whatever could be inferred from it, could be inferred without it as well*. Formally, $|p_a(k) - p_s(k)| = \mathsf{negl}(k)$. This is exactly what the definition of semantic security states.

Quite remarkably, we will show that *semantic security is equivalent to indistinguishability*! We notice that by proving this equivalence we will gain the powerful implications of semantic security while being able to work with the simpler IND-security definition. And this is exactly what we show:

**Theorem 5** *A cryptosystem $(G, E, D)$ is semantically secure if and only if it is* IND*-secure.*

## 8.1 IND-security implies semantic security

Suppose a cryptosystem $(G, E, D)$ is not semantically secure. We wish to show that it is not IND-secure either.

By assumption, there exists a PPT environment Env and adversary $A$ such that for all PPT simulators $S$, $|p_a(k) - p_s(k)| \geq \epsilon$ where $\epsilon$ is non-negligible (i.e., it is inverse-polynomial for infinitely many $k$). So, in particular, let $S$ be as follows:

- $S$: On input $(\alpha, PK)$,
  - Let $c' \leftarrow E(0;\ PK)$.
    Here by 0 we denote an arbitrary fixed message in the domain of our encryption (we assume wlog that such fixed message can be easily obtained from $PK$).
  - Output $z' \leftarrow A(c', \alpha, PK)$.

By assumption, for this $S$ we have $|p_a(k) - p_s(k)| \geq \epsilon$. To understand the above better, we see that we really perform the following experiment:

$$\begin{array}{rcl} (PK, SK) & \leftarrow & G(1^k) \\ (m, R, \alpha) & \leftarrow & \mathsf{Env}(PK) \\ c \leftarrow E(m;\ PK) \quad \text{and} & & c' \leftarrow E(0;\ PK) \\ z \leftarrow A(c, \alpha, PK) \quad \text{and} & & z' \leftarrow A(c', \alpha, PK) \end{array}$$

Then $p_a(k) = \Pr(R(m, z) = 1)$, while $p_s(k) = \Pr(R(m, z') = 1)$, and our assumption says

$$|\Pr(R(m, z) = 1) - \Pr(R(m, z') = 1)| \geq \epsilon \tag{2}$$

We notice that the above almost literally defines a distinguisher between encryption of $m$ and of 0. To formalize this, we now construct the following PPT adversary $B = (B_1, B_2)$ that will break the IND-security of our encryption by making sub-routine calls to Env and $A$:

- $B_1$: On input $PK$,
  - Obtain $(m, R, \alpha) \leftarrow \mathsf{Env}(PK)$.
  - Output $(m, 0, \beta)$, where state information $\beta = (m, R, \alpha)$.
- $B_2$: On input $(\tilde{c}, \beta, PK)$, where $\beta = (m, R, \alpha)$,
  - Obtain $\tilde{z} \leftarrow A(\tilde{c}, \alpha, PK)$.

– If $R(m, \tilde{z}) = 1$, output $\tilde{b} = 0$ (i.e. message is $m$), else $\tilde{b} = 1$ (i.e. message is 0).

The fact the $B = (B_1, B_2)$ break the indistinguishability of $(G, E, D)$ is almost obvious. Indeed, if $b = 0$ (i.e., message is $m$), then $\tilde{c} = c \leftarrow E(m;\ PK)$, $\tilde{z} = z \leftarrow A(c, \alpha, PK)$, and $\Pr(\tilde{b} = b) = \Pr(R(m, z) = 1) = p_a(k)$. Similarly, if $b = 1$ (i.e., message is 0), then $\tilde{c} = c' \leftarrow E(0;\ PK)$, $\tilde{z} = z' \leftarrow A(c', \alpha, PK)$, and $\Pr(\tilde{b} = b) = \Pr(R(m, z) = 0) = 1 - p_s(k)$. Overall

$$\Pr(\tilde{b} = b) = \frac{1}{2} \cdot (p_a(k) + 1 - p_s(k)) = \frac{1}{2} + \frac{1}{2} \cdot (p_a(k) - p_s(k))$$

which is non-negligibly different from 1/2 by Equation (2). This completes the proof that IND-security implies semantic security.

### 8.2 Semantic security implies IND-security

Suppose that our cryptosystem $(G, E, D)$ is not IND-secure, i.e. there exists an adversary $B = (B_1, B_2)$ such that for some non-negligible $\epsilon = \epsilon(k)$,

$$\begin{array}{lll} \Pr[\ b = \tilde{b} \ \mid & (PK, SK) \leftarrow G(1^k); & \\ & (m_0, m_1, \beta) \leftarrow B_1(PK); & \\ & b \leftarrow \{0, 1\}; & \\ & \tilde{c} \leftarrow E(m_b;\ PK); & \\ & \tilde{b} \leftarrow B_2(\tilde{c}, \beta)\ ] & > \frac{1}{2} + \epsilon(k) \end{array}$$

We notice that wlog we can assume that $B_1$ never outputs $m_0 = m_1$. Let us construct an environemnt Env and an adversary $A$ that break the semantic security of $(G, E, D)$ as follows:

- Env: On input $PK$,
  - Obtain $(m_0, m_1, \beta) \leftarrow B_1(PK)$.
  - Let $R$ be the equality relation ($R(x, y) = 1 \iff x = y$), and let adversary's partial information be $\alpha = (m_0, m_1, \beta)$.
  - Output $(m_b, R, \alpha)$, where $b \leftarrow \{0, 1\}$ is a random bit (not given to the adversary!)
- $A$: On input $(c, \alpha, PK)$, where $\alpha = (m_0, m_1, \beta)$,
  - Obtain $\tilde{b} \leftarrow B_2(c, \beta)$.
  - Output $z = m_{\tilde{b}}$.

It is clear that the value $p_a(k)$ for this adversary is exactly the probability of success of $B = (B_1, B_2)$, since the experiments and the success condition ($m_b = m_{\tilde{b}} \iff b = \tilde{b}$ as we assumed $m_0 \neq m_1$) are *exactly the same*. Namely, our environment picked the random bit $b$ for the adversary, but did not reveal it in the partial information $\alpha$. Thus, $p_a(k) > \frac{1}{2} + \epsilon$.

On the other hand, what is the best chances for any algorithm $S$ that does not take $c$ as input? Well, $S$ has to predict a random bit $b$ about which it gets no information at all (remember that Env picked $b$ after $\alpha$ — the input to $S$ — was already computed)! Hence, $p_s(k) = \frac{1}{2}$, which combined with $p_a(k) > \frac{1}{2} + \epsilon$ contradicts the semantic security of $(G, E, D)$.

## 8.3 Other definitions

Twenty years ago, there were no established definitions of security. But people had intuition that "seemed right." The intuition behind the RSA cryptosystem is that, without any a-priori information on the message, it is hard to infer it completely from seeing its encrypted form. As we have stressed before, this intuition of "one-wayness" is much weaker than the intuition behind IND-security.

Yao had another intuition, which had to do with effectiveness of data compression. His intuition is that, if a cryptosystem is secure, then it takes just as many bits to transmit a message, whether I happen to have this message in encrypted form already or not. Namely, the encryption does not help me to further compress the message. From the point of view of an information-theorist, this intuition is very natural, and quite akin to Shannon's ideas about security. It turns out that Yao's definition is equivalent to semantic security (and thus indistinguishability)! This was shown by Micali, Rackoff, and Sloan. Hence, we start to see very convincing evidences that *there is something fundamental in our notion of security, since so many differently looking definitions turned out to be equivalent!*

Notice, it is possible that other notions of security may prove useful, even if they are not comparable to existing ones. In particular, semantic security is not the only remaining candidate (even though it is the most accepted one). The study of useful notions of security is on-going, and if you have any interesting ideas related to it, please be sure to follow up on them!

WORRY ABOUT DEFINITIONS SO MUCH? We focus on *provable* security, i.e., if we claim that a cryptosystem is secure, then we also exhibit a proof of this claim, under some reasonable computational assumption (where "reasonable" is in the eye of the beholder, of course). Historically, there were so many incorrect cryptosystems and definitions of security suggested, that the need for formalism is clear: why to settle for empirical evidence when you can do better? Of course, whether a given formal definition satisfies one's need for security, or comes short of it, is, needless to say, in the eye of the beholder as well. To summarize, a *discussion* of possible definitions and their strength is central, while the *need* for such definitions is obvious.

| CSCI-GA.3210-001<br>MATH-GA.2170-001 | **Introduction to Cryptography** | March 2, 2012 |
|---|---|---|

# Lecture 7

*Lecturer: Yevgeniy Dodis* *Spring 2012*

Last time we proved that if there exists a secure PKE $\mathcal{E}$ to encrypt one bit, then there also exists a secure PKE $\mathcal{E}'$ to encrypt as many bits as we want. In this lecture, we will go further in this direction, searching for the minimal assumption that implies the existence of a indistinguishable (IND) PKE.

Afterwards, we will temporarily leave the construction of our theoretical framework, and we will move to analyzing efficient and practical encryption schemes, based on less general assumptions. In particular, we will present the *ElGamal* PKE, whose various levels of security can be stated in term of two different (but related) assumptions: the *Computational Diffie-Hellman* (CDH) Assumption, and the *Decisional Diffie-Hellman* (DDH) Assumption. We will also study the related problem of *Diffie-Hellman* key exchange.

Finally, we come back to Secret-Key Encryption (SKE) and study notions of security for SKE. First, we define IND-security for one message and give examples of encryptions that provide such security. Unfortunately, one-message security no longer implies multiple message security for SKE. Thus, we define the notion of IND-security for multiple messages. Then, we extend out notion of IND-security to protect against Chosen Plaintext Attack (CPA). We notice that CPA-security is at least as strong and more natural than multiple message security.

## 1 Minimal Assumption For Indistinguishable Encryption

Up to now, we have introduced a few mathematical entities in our theoretical framework, and a significant part of our work has been in realize their connections. For the sake of clarity, in fig. 1 we sketch the cryptographic primitives we have encountered so far, and how they are related to each other: an arrow from a primitive to another, means that the existence of the first implies the existence of the second.

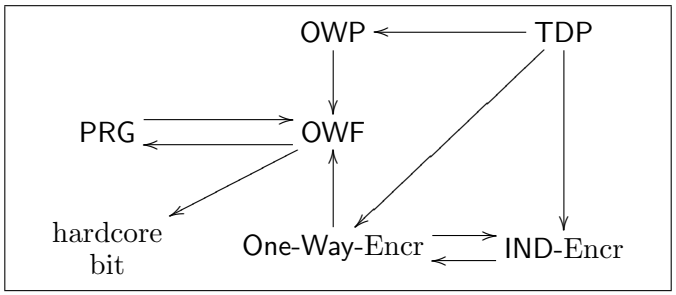



Figure 1: Relations between Cryptographic Primitives

It is worth noticing that the arrow from One-Way-Encryption to IND-Encryption does not mean that each PKE which is One-Way secure is also PKE secure (which is not the

case!). Instead, it means that the existence of a One-Way secure PKE implies the existence of an IND PKE as proved in the following theorem.

**Theorem 1**
*If there exists a* One-Way *secure* PKE $\mathcal{E} = (G, E, D)$ *then there exists an* IND PKE $\mathcal{E}' = (G, E', D')$.

**Proof:** The proof is an application of the One-Time Pad Lemma. By the hypothesis, we have that $\mathcal{E}$ is a One-Way secure PKE, i.e. the encryption function $E$ is a OWF. Using the Goldreich-Levin theorem, we know that a random parity of the input bits of $x$ is a hardcore predicate of $E$:

$$(E(x), r, (x \cdot r \bmod 2)) \approx (E(x), r, b), \quad \text{where } x, r \leftarrow_r M_k, b \leftarrow_r \{0,1\} \qquad (\dagger)$$

Let us define a one-bit PKE $\mathcal{E}' = (G, E', D')$, where to encrypt one bit $b'$ using the randomness $(x, r)$, $E'$ runs $E$ as follows:

$$E'(b'; x, r) = (E(x), r, ((x \cdot r) \bmod 2) \oplus b')$$

The key-generation algorithm for $\mathcal{E}'$ is exactly the same as for $\mathcal{E}$, while the decryption algorithm $D'$ is the straightforward inverse of $E'$:

$$D'(y, r, c) = (((D(y) \cdot r) \bmod 2) \oplus c).$$

To prove that $\mathcal{E}'$ is an IND PKE, we apply the One-Time Pad Lemma, choosing the distributions $X = (E(x), r)$ and $Y = (x \cdot r) \bmod 2$. Therefore, the $(\dagger)$ can be rewritten as $(X, Y) \approx (X, b)$. Thus, by the One-Time Pad Lemma, it holds that $(X, Y \oplus 0) \approx (X, Y \oplus 1)$, that is:

$$E'(0; x, r) = (E(x), r, ((x \cdot r) \bmod 2) \oplus 0) \approx (E(x), r, ((x \cdot r) \bmod 2) \oplus 1) = E'(1; x, r).$$

□

## 2 Efficient Encryption

So far, we have dealt with general constructions and properties of PKEs. However, those constructions, although polynomial, are still inefficient and generally not used in practice. Instead, people typically use optimized schemes based on *specific* number-theoretic assumptions (as opposed to general assumption like the existence of TDPs). In this section we study one of the sysmplest such PKEs, called the *ElGamal* cryptosystem, based on the hardness of the *Diffie-Hellman* Problem, which, in turn, is the basis of a famouse *Diffie-Hellman Key Exchange*.

### 2.1 The Diffie-Hellman Key Exchange

In their seminal paper on Public-Key Cryptography, W. Diffie and M. Hellman proposed the first *Key-Exchange Scheme*, i.e. a scheme to enable two parties, Alice and Bob, to exchange a jointly-selected shared key $s$ to be used in a subsequent, secure communication (for example, as the shared key in a One-Time Pad cryptosystem). The scheme can be sketched as follows:

- Alice and Bob choose a public prime number $p$, and a generator $g$ of the cyclic group $\mathbb{Z}_p^*$ (where $|p| = k$);
- Alice chooses (and keeps secret) an exponent $a \in \mathbb{Z}_{p-1}$, then computes $\alpha = g^a \bmod p$, and finally sends $\alpha$ to Bob;
- Bob chooses (and keeps secret) an exponent $b \in \mathbb{Z}_{p-1}$, then computes $\beta = g^b \bmod p$, and finally sends $\beta$ to Alice;
- as soon as Bob receives the message $\alpha$ from Alice, he computes $s_B = \alpha^b \bmod p$;
- as soon as Alice receives the message $\beta$ from Bob, she computes $s_A = \beta^a \bmod p$.

Clearly, at the end of this protocol, Alice and Bob share the secret key $s = g^{ab} \bmod p$:

$$s_A \equiv \beta^a \equiv (g^b)^a \equiv (g^a)^b \equiv \alpha^b \equiv s_B \pmod{p}$$

Notice that, while Alice or Bob can easily compute $s$ (since they knows $a$ and $b$ respectively), an eavesdropper Eve has to face the much more difficult problem of computing $g^{ab}$ given knowledge $(p, g, g^a \bmod p, g^b \bmod p)$: we will refer to this problem as the *Diffie-Hellman Problem* (DHP).

$$\begin{array}{rl} \text{INPUT:} & p,\ g \text{ generator of } \mathbb{Z}_p^*,\ (g^a \bmod p) \text{ and } (g^b \bmod p) \\ \text{OUTPUT:} & g^{ab} \bmod p \end{array}$$

But how can Eve succeed in her task? Of course, if Eve were able to compute the discrete logarithms in $\mathbb{Z}_p^*$, then she could compute $g^{ab} \bmod p$, by first extracting $a$ from $g^a \bmod p$, and then raising $g^b \bmod p$ to the $a^{th}$ power. It follows that Diffie-Hellman problem is at most as difficult as the more general Discrete Logarithm problem. Of course, this does not imply the equivalence of the two problems: it could certainly be the case that computing discrete logarithms is hard, while solving the Diffie-Hellman problem is possible. This leads to the following assumption:

DEFINITION 1 [Computational Diffie-Hellman (CDH) Assumption over $\mathbb{Z}_p^*$]
$\forall$ PPT algorithm $A$

$$\Pr[A(p, g, g^a, g^b) = g^{ab} \mid p\text{-prime}, |p| = k, a, b \leftarrow \mathbb{Z}_{p-1}, g \text{ generator of } \mathbb{Z}_p^*] = \mathsf{negl}(k)$$

$\diamondsuit$

In other words, the CDH assumption states that "it is hard to *completely* compute $g^{ab} \bmod p$". But this is not enough for key exchange, for the same reason that one-wayness is not enough to guarantee a secure encryption. Indeed, for all we know, the attacker may obtain a lot of partial information about the key $g^{ab}$, which would mean, for example, that it is not safe to use $s = g^{ab}$ as a one-time pad. What we need is that *the agreed key should look indistinguishable from random to Eve.* This leads to the following general definition.

DEFINITION 2 [Passive Security of Key Exchange, semi-formal] We say that a key exchange protocol is *secure against a passive*[1] *observer* if for any PPT attacker $A$ we have $\langle T, s \rangle \approx$

[1] The reason for the term "passive" will become clear very soon.

$\langle T, R\rangle$, where $T$ is the public transcript of the protocol, $s$ is the key agreed upon by Alice and Bob, and $R$ is a completely independent random key of the same length as $s$. $\diamondsuit$

For example, for the DH key exchange the transcript, which is the information that Eve has, consists of $T = \langle p, g, g^a, g^b\rangle$. Thus, in order for the DH key exchange to be passively secure it is necessary and sufficient to make the following stronger assumption, which is called *Decisional Diffie-Hellman (*DDH*) assumption*:

DEFINITION 3 [Decisional Diffie-Hellman (DDH) Assumption in $\mathbb{Z}_p^*$]
$\forall$ PPT algorithm $A$

$$\Big|\Pr[A(p,g,g^a,g^b,g^{ab}) = 1 \mid p\text{-prime}, |p| = k, a, b \leftarrow \mathbb{Z}_{p-1}, g \text{ generator of } \mathbb{Z}_p^*] -$$
$$\Pr[A(p,g,g^a,g^b,g^c) = 1 \mid p\text{-prime}, |p| = k, a, b, c \leftarrow \mathbb{Z}_{p-1}, g \text{ generator of } \mathbb{Z}_p^*\Big| =$$
$$= \mathsf{negl}(k)$$

Equivalently, it is infeasible to distinguish between $g^{ab}$ and $g^c$ given $(p, g, g^a, g^b)$ (we omit $p$ below):

$$(g, g^a, g^b, g^{ab}) \approx (g, g^a, g^b, g^c), \quad \text{where } a, b, c \leftarrow_r \mathbb{Z}_{p-1}$$

$\diamondsuit$

**Lemma 1** *Under the* DDH *assumption in $\mathbb{Z}_p^*$, the DH key exchange is passively secure.*

Unfortunately, as stated, the DDH assumption is false!

**Lemma 2** DDH *assumption is false in $\mathbb{Z}_p^*$.*

**Proof:** The problem is that Eve can learn the quadratic character of $g^{ab} \bmod p$. Recall, the quadratic character $\chi(x)$ for an element $x \in \mathbb{Z}_p^*$ is defined as: $\chi(x) = \begin{cases} 0 & \text{if } x \in QR_p \\ 1 & \text{if } a \notin QR_p \end{cases}$, where $QR_p$ is the subgroup of quadratic residues modulo $p$. Equivalently, it is equal to the least significant bit (LSB) of the discrete logarithm of $x$ base $g$. Either way, it can be efficiently computed by testing wther or not $g^{(p-1)/2} \bmod p \equiv 1$. Thus, from $g^a$ Eve can compute $\mathsf{LSB}(a)$, from $g^b$ — $\mathsf{LSB}(b)$, and then since $(p-1)$ is even

$$\mathsf{LSB}(ab \bmod (p-1)) = (ab \bmod (p-1)) \bmod 2 = ab \bmod 2 = \mathsf{LSB}(a) \oplus \mathsf{LSB}(b)$$

Put differently, $g^{ab}$ is a quadratic character iff either both $g^a$ abd $g^b$ are, or none are. But this means that given the public transcript, Eve can compute the quadratic character $\chi(s)$ of the agreed key $s$, which means that $s$ is distinguishable from ranndom (since a random $x$ has a random quadratic character which cannot be predicted). In other words, the scheme leaks some information about the key. $\square$

It is common practice to fix this problem via an *ad hoc* solution, that consists in imposing a specific structure on the prime $p$. Let us assume that $p = 2q+1$, with $p$ and $q$ both primes (such a prime $p$ is called *strong prime*). Then consider the subgroup $G$ of $\mathbb{Z}_p^*$ made up by

all the quadratic residues modulo $p$: in other words $G \doteq QR_p$ . It is known that the order of this subgroup is $\frac{p-1}{2} = q$, i.e. $QR_p$ has prime order for our choice of $p$. Moreover, if $h$ is a generator of $\mathbb{Z}_p^*$, then $g = h^2 \bmod p$ is a generator of $G$. Finally, $(G, \cdot)$ is isomorphic to $(\mathbb{Z}_q, +)$, since $g^a \cdot g^b \equiv g^{(a+b) \bmod q} \pmod p$. Indeed, by definition of *generator* and of *order* of a group, it holds that $g^q \equiv 1 \pmod p$. Now, writing $a + b = c \cdot q + (a + b) \bmod q$ for some $c$, we can make the following considerations:

$$g^a \cdot g^b \equiv g^{(a+b)} \equiv g^{c \cdot q} \cdot g^{(a+b) \bmod q} \equiv \left(g^q\right)^c \cdot g^{(a+b) \bmod q} \equiv 1^c \cdot g^{(a+b) \bmod q} \pmod p$$

Finally:

$$g^a \cdot g^b \equiv g^{(a+b) \bmod q} \pmod p .$$

We can now modify the Diffie-Hellman Key-Exchange protocol, so that the random values $a$ and $b$ chosen by Alice and Bob are drawn from $\mathbb{Z}_q$ (instead of $\mathbb{Z}_{p-1}$), and the whole computation is performed in $G$ (instead of $\mathbb{Z}_p^*$).

The reason why these changes are effective is that now it is *always the case* that $g^a \bmod p$, $g^b \bmod p$ and $g^{ab} \bmod p$ are quadratic residues, and so what Eve can learn trough the considerations above (namely, the quadratic character of $g^{ab} \bmod p$) is already publicly know.

Of course, this "fix" does not a-priori *guarantee* that no other leakage of partial information is present by some other means. Nevertheless, extensive attempts to find other attacks on this modied scheme have failed, and, therefore, people came up with the following variants of the CDH and DDH assumptions in this new, slightly modified setting. Both variants are believed to be secure for a large enough security paprameter.

Definition 4 [Computational Diffie-Hellman (CDH) Assumption in $QR_p$]
$\forall$ PPT algorithm $A$

$$\Pr[A(g^a, g^b) = g^{ab} \mid p = 2q + 1, |p| = k, p, q \text{ primes}, a, b \leftarrow_r \mathbb{Z}_q, g \text{ generator of } G] = \mathsf{negl}(k)$$

*(where it is omitted that $A$ also knows the public modulo $p$ and the generator $g$.)* $\diamondsuit$

Definition 5 [Decisional Diffie-Hellman (DDH) Assumption in $QR_p$]
$\forall$ PPT algorithm $A$

$$\Big| \Pr[A(g^a, g^b, g^{ab}) = 1 \mid p = 2q + 1, |p| = k, p, q \text{ primes}, a, b \leftarrow_r \mathbb{Z}_q, g \text{ generator of } G] -$$
$$\Pr[A(g^a, g^b, g^c) = 1 \mid p = 2q + 1, |p| = k, p, q \text{ primes}, a, b, c \leftarrow_r \mathbb{Z}_q, g \text{ generator of } G] \Big| =$$
$$= \mathsf{negl}(k)$$

*(where, again, it is omitted that $A$ also knows the public modulo $p$ and the generator $g$.)* $\diamondsuit$

Similarly to Lemma 1, we get

**Lemma 3** *Under the* DDH *assumption in $G = QR_p$, the modified DH key exchange is passively secure.*

We still have two more problems:

KEY IS NOT A "BIT-STRING". As it is defined, the DH key exchange outputs a key $s = g^{ab} \bmod p$ which looks like a random element of the group $G = QR_p$. As such, it appears that it cannot be directly used for a one-time pad or other general purposes. In other words, ideally we want to output a usual bit-string as our key. There are several ways out of this. One very general way is to usilize some "magic function" $H$ from $G$ to bit-strings of some length $\ell$ (roughly, $\ell \approx k$) with the property that a random element of $G$ gets mapped to an almost random element of $\{0,1\}^\ell$. Turns out such general functions exist and called *randomness extractors*.

Here we describe a simple construction of such $H$ for our specific choice of $G = QR_p$. Actually, let us first map from $G$ to $\mathbb{Z}_q$ Since $p = 2q + 1$ and $q$ is prime, $q$ is odd. Hence, $q = 2k + 1$, and substituting we obtain $p = 4k + 3$, or alternatively $p \equiv 3 \pmod 4$. Then we know that any $s \in \mathbb{Z}_p^* = \{1 \dots 2q\}$, precisely only one among $s$ and $-s$ is a quadratic residue. Similarly, precisely one of $s$ and $-s$ is between 1 and $q$. Thus, we have a bijection between $QR_p$ and $\mathbb{Z}_q$: map any $s \in QR_p$ either to $s$, if $s \le q$, or to $p - s$ is $s > q$.

Still, we only get a radnom number $t$ from 1 to $q$, and $q$ is not a precise power of 2. However, to get a bit-string which is very close to uniform we can simply truncate a few bits of $t$. Concretely, if $\ell$ is an integer s.t. $2^\ell < q$, then the "variation distance" between a random $\ell$-bit string and our "extracted" string is less than $2^\ell/q$, so by dropping roughly 80 bits we get a nearly perfect random string.

ACTIVE ATTACKER. The second problem is that there is no way for Bob to know who is the sender of a message. In particular, the protocol we have allows Eve to mount the so called *person-in-the-middle* attack:

- Alice chooses (and keeps secret) an exponent $a \in \mathbb{Z}_q$, then computes $g^a \bmod p$, and sends it to Bob;
- Eve intercepts $g^a \bmod p$ (thus preventing Bob from getting it), and sends $g^{a'} \bmod p$ to Bob, for a random $a'$ of her choice;
- Bob chooses (and keeps secret) an exponent $b \in \mathbb{Z}_q$, then computes $g^b \bmod p$, and sends it to Alice;
- Eve intercepts $g^b \bmod p$ (thus preventing Alice from getting it), and sends $g^{b'} \bmod p$ to Alice, for a random $b'$ of her choice;
- when Bob receives the message $g^{a'} \bmod p$, he (erroneously) assumes that it comes from Alice, and thus he sets $s_B = (g^{a'})^b \bmod p = g^{a'b} \bmod p$;
- when Alice receives the message $g^{b'} \bmod p$, she (erroneously) assumes that it comes from Bob, and thus she sets $s_A = (g^{b'})^a \bmod p = g^{ab'} \bmod p$;
- Eve can easily compute both $s_A = (g^a)^{b'} \bmod p = g^{ab'} \bmod p$ and $s_B = (g^b)^{a'} \bmod p = g^{a'b} \bmod p$.

As a consequence of the attack, at the end of this run of the protocol, Alice and Bob happily enjoy their "secure" connection, exchanging their love messages and saying nasty

things about Eve, but $\ldots s_A \neq s_B$, and so actually they are talking trough Eve, who can not only learn everything, but also manipulate the communication. Notice also that the attack is generic: *any* protocol where Alice and Bob have no way to "authenticate" each other allows Eve to have 2 disjoint conversations with them just like below. Thus, to avoid this attack, a key point is to add *Authentication* to the basic Key-Exchange Scheme, i.e. provide some mechanisms that enables the receipient of a message to be sure about the identity of the sender. This is typically achieved by means of *Digital Signature*, which will be discussed later. When we do it, we will be able to get key exchange protocols resisting an *active* rather than passive attacker. But, for now, we need to assume that Eve is passive and merely listens to the conversation when trying to deduce information about the key $s$.

## 2.2 Back to Encryption: The ElGamal Cryptosystem

Now, we describe the ElGamal Cryptosystem, which exploits the hardness of the DHP to achieve a reasonable level of security quite efficiently.

1. On input $1^k$, the *key-generation algorithm* Gen chooses a strong prime $p = 2q + 1$ along with a generator $h$ of $\mathbb{Z}_p^*$, and sets $g = h^2 \bmod p$. Afterwards, $G$ takes a random element $x \in \mathbb{Z}_q$ and sets $y = g^x \bmod p$. Finally, $G$ outputs $(PK, SK, M_k)$, where $PK = (p, g, y)$, $SK = x$, and the message space is $M_k = G = QR_p$.

2. Given a message $m \in M_k$ and some randomness $r$, the *encryption algorithm* $E_{PK}$ outputs the ciphertext $(s, t) = (g^r, y^r \cdot m)$, where $PK = (p, g, y)$.[2] Notice that, since $y = g^x$, the ciphertext is actually of the form $(g^r, g^{xr} \cdot m)$, although, of course, this form is "hidden', and Eve can only see $(s, t)$.

3. In order to decrypt a ciphertext $(s, t)$, given the private key $SK = x$, the decryption algorithm $D$ first recovers the quantity $s^x = g^{rx} = y^r$, and then 'simplifies' it out from $t$, computing $t \cdot (s^x)^{-1} = t \cdot (y^r)^{-1} = y^r \cdot m \cdot (y^r)^{-1} = m$.

4. In attacking the cryptosystem, the adversary Eve knows the public key $PK = (p, g, y = g^x)$ and a ciphertext $(s, t) = (g^r, y^r \cdot m)$, corresponding to $m$: overall, Eve's knowledge can be represented as $(g^r, g^x, g^{xr} \cdot m)$.

Comments.

- Both the encryption and the decryption algorithms of this scheme are fairly efficient, since, in each invocation, at most two modular exponentiations are required.

- Each ciphertext produced by the encryption algorithm is exactly twice as long as the plaintext. This space overhead is induced by the presence of some randomness in the encryption algorithm — which is unavoidable in any PKE that 'aims' to fulfil a notion of security that is stronger than One-Way security: as we noticed in previous lectures, a share of non-determinism is necessary to be able to send more than once the same message, in such a way that Eve cannot recognize that two different ciphertexts are related to each other.

[2] Where omitted, the computation is assumed modulo $p$.

- As it is defined, the ElGamal Scheme allows us to encrypt only messages that are elements of the group $G$. Usually, the messages we want to encrypt are not numbers, although they can easily be mapped onto numbers in some specific range (say, from 1 to $q$): but how can we force such numbers to be quadratic residues? We simply do the inverse mapping to the one described in the previous section, when we mapped a quadratic residue in $G = QR_p$ to an element of $\mathbb{Z}_q$. Namely, for our choice of $p$ we showed that for any $m \in \mathbb{Z}_q$, precisely one of $m$ and $-m$ belongs to $G$, so we simply apply the ElGamal encryption to either $m$ or $-m$, whichever is the quadratic residue.

  Of course, we can turn this around, and directly encrypt $m \in \mathbb{Z}$, by changing the encryption to $(g^r, H(y^r)+m \bmod q)$, where $H$ is the mapping described in the previous section: $H(z) = z$ if $z < q$ and $H(z) = -z$ if $z \geq q$. We will see, however, that this way we lose the homomorphic properties of ElGamal encryption that we describe next.

Homomorphic Properties. ElGamal Encryption has several very nice properties which we describe here informally.

- ElGamal encryption is *homomorphic*. Given the public key and two encryptions $(s_0, t_0)$ and $(s_1, t_1)$ of some *unknown* messages $m_0$ and $m_1$, one can efficiently compute an encryption $(s_0 s_1 \bmod p, t_0 t_1 \bmod p)$ of $m_0 m_1 \bmod p$. Indeed, if $r_0$ and $r_1$ are the coins used in the above encryptions, we have

$$(s_0 s_1 \bmod p, t_0 t_1 \bmod p) = (g^{r_0+r_1}, y^{r_0+r_1} \cdot (m_0 m_1)) = E(m_0 m_1;\ r_1 + r_2)$$

  More generally, for any integers $i$ and $j$ one can compute encryption of $m_0^i m_1^j \bmod p$ in a similar manner, by rasing computing $(s_0^i s_1^j, t_0^i t_1^j)$.

- ElGamal encryption is *blindable*. Given the public key, some encryption $(s, t)$ of some *unknown* message $m$, and any value $m' \in G$, one can efficiently compute an encryption of $mm' \bmod p$. Indeed, this follows from the momomorphic property above, or can be seen directly by computing $(s, t \cdot m')$. Moreover, a *random* encryption of $m \cdot m'$ can be obtained by picking a random $r' \in \mathbb{Z}_q$ and computing $(s \cdot g^{r'}, t \cdot y^{r'} \cdot m')$. The verification of this fact is similar to the above.

- ElGamal encryption is *re-randomizable*. Given the public key and an encryption $(s, t)$ of some *unknown* message $m$, one can compute a fresh *random* encryption of the same $m$ which is identical to the process of really encrypting *known* $m$ from scratch. Simply apply the previous scheme to $m' = 1$, or, directly, pick random $r'$ and compute $(s \cdot g^{r'}, t \cdot y^{r'})$.

These homomorphic properties, by themself, are neither good nor bad: it depends upon the scenario at hand. Consider, for example, the case of a centralized authority $C$ that is able to decrypt messages for others, but should not be allowed to learn the content of those messages. At first glance, this may seem hopeless, but the "blindable" property comes in help: instead of submitting the actual ciphertext, one can choose a random message $m'$ and send to $C$ a "blinded" ciphertext for $m \cdot m'$ (without knowng $m$, of course!). When $C$ decrypts, it cannot understand the content (since the randomness of $m'$ makes $mm'$

random!), but the other party can recover $m$ by simply dividing out by $m'$ (which it chose and knows).

On the other hand, in the setting of an auction, the "blindable" property would allow a participant to "cheat" by doubling the price offered by another participant, even when the auction bets are being encrypted with the ElGamal PKE of the auctioneer.

Relation to Key Encapsulation and Key Exchange. First, we notice that the ElGamal encryption also follows the general key encapsulation paradigm we studied earlier, except the XOR operation is replaced by multiplication in $G$. In particular, we get the following key encapsulation mechanism $(\mathsf{Gen}, E, D)$.

- Gen is exactly the same as for the ElGamal cryptosystem. It chooses a strong prime $p = 2q+1$ along with a generator $h$ of $\mathbb{Z}_p^*$, and sets $g = h^2 \bmod p$. Afterwards, $G$ takes a random element $x \in \mathbb{Z}_q$ and sets $y = g^x \bmod p$. Finally, $G$ outputs $(PK, SK, M_k)$, where $PK = (p, g, y)$, $SK = x$, and the message space is $M_k = G = QR_p$.
- The key encapsulation algortihm $KE(PK)$. It chooses a random $r$ and sets key $s = y^r \bmod p$ and ciphertext $\psi = g^r \bmod p$.
- The key decapsulation algortihm $KD(\psi; SK)$ outputs $s = \psi^x \bmod p$.

The correctness is tested exactly as in the the DH key exchange: $\psi^x = g^{rx} = y^r$. And this is more than a coincidence. Indeed,

**Observation 2** *Any two-round key exchange secure against a passive attacker yeilds a key encapsulation mechanism which is* CPA*-secure, and vice versa.*

**Proof:** First, we do the easier direction. Given any KEM, construct two-round key exchange as follows.

- In the first round, Alice choses $(PK, SK) \leftarrow \mathsf{Gen}(1^k)$ and sends $PK$ to Bob.
- In the second round, Bob computes $(s, \psi) \leftarrow KE(PK)$, and sends ciphertext $\psi$ to Alice. Bobn's key is $s$.
- Alice recovers $s = KD_{SK}(\psi)$.

The converse is also very similar, just view the first message as the public key $PK$ of the KEM, the second message from Bob — as the ciphertext, Alice's coins for the first message as $SK$, and Bob's key as the value $s$. □

Applied to the DH key exchange, we *exactly get the above* KEM! Indeed, the public key is $g^x$, and encryption of the key $g^{rx}$ is indeed $g^r$. Combining with Lemma 3, we get

**Lemma 4** *Under the* DDH *assumption in $QR_p$, the ElGamal* KEM *is* CPA*-secure.*

Security of the ElGamal Cryptosystem. We can now use the following generalization of the one-time pad lemma to deduce a similar security for ElGamal encryption:

**Lemma 5 (Generalized One-Time Pad Lemma)** *Let $(G, \cdot)$ be a group, and $R$ denote the uniform distribution over $G$. For all the distributions $X, Y$ (not necessarily independent), if $(X, Y) \approx (X, R)$, then for all $m_0, m_1 \in G$, we have $(X, Y \cdot m_0) \approx (X, Y \cdot m_1)$. More generally, the "CPA-analog" of this result is true as well, where the PPT attacker is allowed to choose $m_0, m_1$ based on $X$.*

**Proof:** Just substitute $\cdot$ for $\oplus$ in the proof of the One-Time Pad Lemma (see Lecture 6). We leave the "CPA-analog" as an exercise. □

Using this result, we immediately get

**Theorem 3** *Under the DDH assumption, the ElGamal cryptosystem is an IND secure PKE.*

Finally, we remark that we can also prove a weaker one-time secuirty of ElGamal under a weaker CDH assumption.

**Theorem 4** *Under the CDH assumption, the ElGamal cryptosystem (or KEM) is a One-Way secure PKE.*

**Proof:** For the sake of contradiction, let us assume that the ElGamal cryptosystem is not a One-Way secure PKE:
$\exists$ PPT algorithm A such that:

$$\Pr[A(g^x, g^r, g^{xr} \cdot m) = m \mid p = 2q + 1, |p| = k, p, q \text{ primes}, x \leftarrow_r \mathbb{Z}_q, g \text{ generator of } G] = \epsilon$$

for some non-negligible $\epsilon$.

Using a reduction approach, we now construct an algorithm $A'$ which, given black-box access to the algorithm $A$, can efficiently solve the DHP with non-negligible probability, contradicting the hypothesis. On input $(g^x, g^r)$, for random $x$ and $r$ in $\mathbb{Z}_q$, $A'$ chooses $c \leftarrow_r \mathbb{Z}_q$ and runs $A(g^x, g^r, g^c)$: with probability $\epsilon$, $A$ will decrypt the 'ciphertext' $g^c$ and thus will output $\tilde{m} = g^c \cdot (g^{xr})^{-1}$. At this point, $A'$ computes $g^c \cdot \tilde{m}^{-1}$ as its guess for $g^{xr}$. Therefore, with probability roughly $\epsilon$, $A'$ will successfully solve the Diffie-Hellman Problem. □

**Remark 1** *Quite interestingly, based on two different assumptions, it is possible to show that the* same *construction fulfil two different notions of security. Moreover, the assumption made and the security obtained are well-coupled: assuming that it is difficult to completely solve the DHP leads to the difficulty of completely breaking the ElGamal PKE; assuming that it is infeasible to learn anything useful about the solution of the DHP leads to the difficulty of learning anything useful about messages encrypted with the ElGamal PKE.*

## 2.3 Other Efficient Encryption Schemes

There are other IND-CPA-secure encryption schemes. In the homework we will study the original *Goldwasser-Micali (GM) cryptosystem*, which allows one to encrypt messages bit-by-bit (so it's not very efficient) and is secure under the quadratic residuosity assumption. The GM scheme is also homomorphic over $\mathbb{Z}_2$. A more recent efficient cryptosystem is called

a *Paillier* encryption, and is based on a stronger decisional variant of the RSA assumption which we will not state here. This schemes directly allows one to encrypt relatively large messages, but taken over group $\mathbb{Z}_n$, where $n$ is a certain RSA-type modulus. Similarly to ElGamal, it is also homomorphic. Unlike ElGamal, it is homomorphic under modular *addition* rather than multiplication, so it's more convenient to use in many applications.

Finally, we will come back to public-key encryption when studying a stronger attack called chosen *ciphertext* attack (CCA), as opposed to the CPA attack we studied so far.

# 3 Secret-Key Encryption

We now return back to symmetric-key encryption. First, we remind ourselves the definition of Secret-Key encryption. (Notice the similarity to the definition of Public-Key encryption given in Lecture 6.)

DEFINITION 6 A Secret-Key encryption (SKE) is a triple of PPT algorithms $\mathcal{E} = (G, E, D)$, where

1. $G(1^k)$ as before outputs $(PK, SK, M_k)$, $PK, SK, M_K$ being a public key, secret key, and message space respectively. $k$ is the security parameter.
2. $E(m; PK, SK, r)$ is the encryption algorithm that outputs ciphertext $c$. Note the difference: the input includes the secret key $SK$. The presence of the secret key contrasts SKE and PKE.
3. $D(c; SK)$, which is usually deterministic, outputs a decrypted message $\tilde{m} \in \{\mathsf{invalid}\} \cup M_k$.

As before, we require the correctness property: $\forall m \in M_k, \tilde{m} = m$. That is, if $m$ was correctly encrypted using appropriate secret and public keys. ♢

For simplicity, we will omit $PK$ and incorporate all $PK$ information into $SK$. The use of $PK$ may yield more efficient implementations because it may help reduce the size of the secret storage. Omitting the public key, however, will not cause any loss of generality in our discussion below. Thus, we often denote $SK = s$ and write $c \leftarrow E_s(m)$, $m \leftarrow D_s(m)$.

Now that SKE is defined, we will attempt to formally define the security notions for SKE.

## 3.1 One-Message IND-Security Against PK-Only Attack

We start with the simplest definition of security. Our first objective is to obtain security for one message only with adversaries unaware of any non-public information and unable to choose plaintext.

DEFINITION 7 SKE is one-message IND-secure against $PK$-only attack iff for any two messages, their ciphertexts are polynomially indistinguishable. $\forall m_0, m_1 \in M_k, E_s(m_0) \approx E_s(m_1)$. ♢

**Example 1.** One-Time Pad satisfies the above definition of security. Formally, $G(1^k) \rightarrow (\perp, R, M_k)$, where $M_k = \{0,1\}^k$, $R \in M_k$ is a random string.
Let $E_R(m) = m \oplus R$ and $D_R(c) = c \oplus R$. It is easy to see that $E_R(m_0) \equiv E_R(m_1)$, since both define truly random strings.

**Example 2.** One can reduce the size of the secret key in Example 1 by generating a pseudo-random string of size $k$ using a PRG $G : \{0,1\}^k \rightarrow \{0,1\}^n$ where $n = p(k)$. Because $G(x) \approx \{0,1\}^{p(x)}$, we can use the one-time pad lemma to conclude that this more efficient version is one-message IND-secure.

The two examples above are no longer secure if more than one messages are to be encrypted. Indeed, $m_0 \oplus m_1 = c_0 \oplus c_1$ can be obtained by an adversary should he intercept ciphertexts $c_0$ and $c_1$ for $m_0$ and $m_1$. Therefore, we are in need of a better definition of IND-security for SKE. This is in contrast to the PKE, where one message security implied multiple message security.

### 3.2 SKE IND-Security With Respect To Multiple Messages

DEFINITION 8 SKE is IND-secure with respect to multiple messages against $PK$-only attack iff $\forall t = \mathsf{poly}(k), \forall m_0^1, m_0^2, ..., m_0^t$ and $m_1^1, m_1^2, ..., m_1^t \in M_k$,

$$E_s(m_0^1) \circ E_s(m_0^2) \circ \ldots \circ E_s(m_0^t) \approx E_s(m_1^1) \circ E_s(m_1^2) \circ \ldots \circ E_s(m_1^t)$$

$\diamondsuit$

In particular, this definition suggests that no information can be inferred about any of the $m_i$'s as long as the number of transmitted messages is polynomial in $k$, the security parameter. Applying the hybrid argument, it can be shown that the following definition is equivalent to Definition 8.

DEFINITION 9 SKE is IND-secure with respect to multiple messages against $PK$-only attack iff $\forall t = \mathsf{poly}(k), \forall i \leq t, \forall m^1, m^2, \ldots, m^{i-1}, m_0^i, m_1^i, m^{i+1}, \ldots, m^t \in M_k$,

$$\begin{aligned} &E_s(m^1) \circ \ldots \circ E_s(m^{i-1}) \circ \quad E_s(m_0^i) \quad \circ E_s(m^{i+1}) \ldots \circ E_s(m^t) \approx \\ &E_s(m^1) \circ \ldots \circ E_s(m^{i-1}) \circ \quad E_s(m_1^i) \quad \circ E_s(m^{i+1}) \ldots \circ E_s(m^t) \end{aligned}$$

$\diamondsuit$

Notice, the last definition can be viewed the following way. The adversary chooses messages $m^1, m^2, \ldots, m^{i-1}, m^{i+1}, \ldots, m^t$, as well as a pair of messages $m_0^i, m_1^i$. Then the adversary gets to see the encryptions of the messages that it chose, and the encryption of $m_b^i$ for a random bit $b$. The adversary then has to guess the bit $b$. Notice, since the adversary is unable to compute any of the encryptions by itself (unlike was possible in the PKE), it essentially means that $A$ has *oracle access to the encryption oracle* $E_s(\cdot)$! Except this oracle access is "non-adaptive": all the messages have to be chosen at the beginning, and all the encryptions have to be given. From this point of view, it seems more natural to not place this unnatural restriction, and give $A$ complete oracle access to $E_s(\cdot)$. Namely, at any point during its run, $A$ can ask the oracle to encrypt any message $m$, and will get back

$E_s(m)$. At some point $A$ chooses two messages $m_0$ and $m_1$. Then one of them $m_b$, will be encrypted, and $\tilde{c} \leftarrow E_s(m_b)$ will be given to $A$ (for random unknown $b$). At this point $A$ can again ask the oracle to encrypt a bunch of messages. Finally, $A$ tries to predict $b$ correctly. This is exactly the *chosen plaintext attack* (CPA). It also explains why we used the same terminology on the public-key setting: the explicit encryption oracle was implicitly given to the adversary by means of the public key $PK$!

We address IND-security against CPA in the next section.

### 3.3 SKE IND-Security Against Chosen Plaintext Attack (CPA)

The following definition summarizes what is said above. We denote by $A^{E_s}$ the adversary who is given oracle access to $E_s(\cdot)$, as explained above.

DEFINITION 10 SKE is IND-secure against CPA iff $\forall \mathsf{PPT} B = (B_1, B_2)$,

$$\Pr[\, b = \tilde{b} \;\mid\; \begin{array}{l} s \leftarrow G(1^k); \\ (m_0, m_1, \beta) \leftarrow B_1^{E_s}(1^k); \\ b \leftarrow \{0,1\}; \\ \tilde{c} \leftarrow E_s(m_b); \\ \tilde{b} \leftarrow B_2^{E_s}(\tilde{c}, \beta); \,] \leq \frac{1}{2} + \mathsf{negl}(k) \end{array}$$

$\diamondsuit$

To reiterate, this means that even if the adversary has an encryption oracle who will encrypt any message the adversary wants (without revealing the secret key), and if the adversary is free to produce any two messages and encrypt them using the oracle, and use any information thus obtained, including the ciphertexts of the two chosen messages, the adversary will still have a negligible advantage in guessing which of the two messages was encrypted by the experimenter.

Now we indicate that Definition 10 is at least as strong as Definition 8 (in fact, can be shown to be strictly stronger), i.e. a SKE that is IND-secure againts CPA can be safely used to transmit multiple messages.

**Lemma 6** *If* SKE $E$ *is* IND-CPA*-secure, then* SKE $\mathcal{E}'$ *with encryption function* $E'_s(m_0, \ldots, m_t) = E_s(m_0) \circ \ldots \circ E_s(m_t)$*, is also* IND-CPA*-secure, for any* $t = poly(k)$*.*

This result is proven using the hybrid argument in a completely identical manner than a similar result for the PKE was shown. Indeed, having oracle access to the encryption function allows the adversary to prepare encryptions of $m_1^0, \ldots, m_{i-1}^0, m_{i+1}^1, \ldots, m_t^1$ which are needed for the hybrid argument. The inability to make such encryptions is exactly the reason the proof fails in general for one-message IND-secure SKE's.

| CSCI-GA.3210-001<br>MATH-GA.2170-001 | **Introduction to Cryptography** October 21, 2008 |
|---|---|

Lecture 8

*Lecturer: Yevgeniy Dodis* *Spring 2012*

In this lecture we build some IND-CPA-secure schemes. Our initial schemes, based on iterating a PRG, are *stateful*. We then turn to the question if there exists a stateless IND-CPA-secure SKE.

# 1 Building CPA-Secure SKE: Stream Ciphers

No CPA-secure encryption can be deterministic, since the adversary $B_2$ can always ask its oracle to re-encrypt $m_0$ and $m_1$. There are two ways out. One way is to make the encryption probabilistic. We will come back to this later. For now, we explore the second way: make the encryption *stateful*. Namely, every time the encryption/decryption operations are performed, the secret key (so called *state*) will change as well.

DEFINITION 1 Stateful SKE is a usual SKE with the following modifications. Encryption and decryption functions $E$ and $D$ additionally output the new value of the secret key $s$, which be used for subsequent encryption/decryptions. Formally, $E(m; s)$ ouputs a pair $(c, s')$, where $c$ is the ciphertext and $s'$ is the new state (secret key). Similarly, $D(c; s)$ ouputs a pair $(m, s')$, where $m$ is the plaintext and $s'$ is the new state (secret key). We require that the decryption and encryption are always in sink: a decryption of $c$ must be performed after every encryption of $m$. $\diamondsuit$

We must also mention that in the definition of CPA-secure SKE, oracle access must keep state as well. This is so because the policy of updating the secret key should be considered public as a part of the algorithm's encryption and decryption functions.

We notice that having state eneables us to possibly have *determinisitc stateful* schemes, as we illustrate.

**Example 1: Blum-Micali Generator For One Bit.** CPA-secure encryption of multiple messages can be achieved using the following stateful algorithm.
Let $f : M_k \to M_k$ be a OWP and $h : M_k \to \{0, 1\}$ be a hardcore bit of $f$.
Encryption function: $E_s(b) \to (c = h(s) \oplus b, s \leftarrow f(s))$.
Decryption function: $D_s(c) = (m = h(s) \oplus c, s \leftarrow f(s))$.

In other words, we simply keep applying the Blum-Micali PRG and using each successive bit as the next one-time pad. The CPA-security of this scheme follows immediately from the one-time pad lemma and the fact that Blum-Micali generator is a PRG. In fact, even if we reveal the adversary the entire state (secret key) after we encrypt the challenge bit $b$, we proved that the adversary cannot predict $b$. Thus, this method is *forward-secure*. Loosing the current secret key protects all the previous encryptions.

Notice also that 1-bit limitation is not an issue. Simply generate more bits to encrypt longer message, and use these bits as a longer one-time pad. This is because we showed

the closure of CPA-secure (as opposed to the one-time secure) symmetric encryption under composition.

**Example 2: Using Any PRG.** Previous construction could be viewed as using the Blum-Micali generator $G(x) = G'(x) \circ f^n(x)$ (see Lecture 5 for notation). In fact, if we write $G(x) = G_1(x) \circ G_2(x)$, where $G : \{0,1\}^k \to \{0,1\}^{n+k}$, and $|G_1(x)| = n$, $|G_2(x)| = k$, we get the that the above construction is a special case of the following more general construction, which works for *any* PRG $G$.

Let $s \leftarrow \{0,1\}^k$ be the initial secret.
To encrypt $m \in \{0,1\}^n$ having current state $s$, output $c = m \oplus G_1(s)$, and update $s = G_2(s)$.
To decrypt $c \in \{0,1\}^n$ having current state $s$, output $m = c \oplus G_1(s)$, and update $s = G_2(s)$.

Again, CPA-security of this stateful construction follow quite easily from the one-time pad lemma, since $G_1(s) \circ G_1(G_2(s)) \circ \ldots \circ G_1(G_2(\ldots G_2(s)\ldots))$ was shown to be a PRG in the previous lectures. In fact, since the PRG above is forward-secure, we get that our general scheme is forward as well (i.e., loosing current state does not compromise previous encryptions).

**Example 3: Using DDH.** This is a special case of Example 2 above, which is obtained by noticing that the DDH assumption immediately gives rise to a new and effiicent PRG! Indeed, let us assume that the prime $p = 2q+1$ and the and generator $g$ of the subgroup $\mathbb{G}$ or quadratic residues modulo $p$ are fixed and public. Now consider the following function $G : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G} \times \mathbb{G} \times \mathbb{G}$ (recall from the last lecture that $\mathbb{Z}_q$ is isomorphic to $\mathbb{G}$, and also we can easily map between them, so one can view $\mathbb{G}$ as going from $Z_q^2$ to $Z_q^3$):

$$G(a,b) = \langle g^a, g^b, g^{ab} \rangle$$

The DDH assumption states that this $G$ is indeed a PRG (with output indistingushable from $\langle g^a, g^b, g^c \rangle$ for random $a, b, c$). This gives an efficient PRG which expands its random input by 50%, going from $2k$ to $3k$ bits.

Even more optimized, let us fix another random generator $h$ of $\mathbb{G}$ (i.e., one can think of $h = g^a$ for a random $a$ which is chosen once and for all). Now define length-doubling $G : \mathbb{Z}_q \to \mathbb{G} \times \mathbb{G}$ (or, alternatively, from $\mathbb{Z}_q$ to $\mathbb{Z}_q^2$) by

$$G(b) = \langle g^b, h^b \rangle$$

Once again, writing $h = g^a$ for a random (unknown) $a$, the attacker's view $\langle g, h, G(b) \rangle \equiv \langle g, g^a, g^b, g^{ab} \rangle \approx \langle g, g^a, g^b, g^c \rangle$, implying that $G$ is a PRG which goes from $k$ to $2k$ bits. As it turns out, this construction works even if $h = g^a$ is fixed "forever" and only new $b$ is chosen for every fresh invocation of $G$.

**Lemma 1** *For any polynomial $t = t(k)$, the DDH assumption implies that, for random $a, b_1, \ldots, b_t, c_1, \ldots, c_t \leftarrow \mathbb{Z}_q$, we have*

$$\langle g, g^a, g^{b_1}, g^{ab_1}, \ldots g^{b_t}, g^{ab_t} \rangle \approx \langle g, g^a, g^{b_1}, g^{c_1}, \ldots g^{b_t}, g^{c_t} \rangle \quad (1)$$

**Proof:** Given a tuple $\langle g, g^a, g^b, g^c \rangle$, where $c$ is either $ab \bmod q$ or random, for $j = 1 \dots t$, pick random $d_j, e_j \leftarrow \mathbb{Z}_q$ and define

$$g^{b_j} = (g^b)^{d_j} g^{e_j}; \quad g^{c_j} = (g^c)^{d_j} (g^a)^{e_j}$$

Notice, $g^{b_j}$ and $g^{c_j}$ can be computed efficiently from $g^a, g^b, g^c$ and $d_j, e_j$. Mathematically, however, if we write $c = ab + z$, where $z = 0$, when $c = ab$, and random, when $c$ is random, we have

$$b_j = bd_j + e_j \bmod q; \qquad c_j = (ab + z)d_j + ae_j = zd_j + a(bd_j + e_j) = zd_j + ab_j$$

Notice, since $e_j$ is random, all values $b_j = bd_j + e_j$ are random and independent of each other, even conditioned on known $d_j$. Moreover, if $z = 0$ (DDH-tuple), we always have $c_j = ab_j$, meaning that we get precisely the left hand side of Equation (1), for random $b_j$. On the other hand, when $z$ is random (non-DDH typle), with high probability $z \neq 0$, and all the values $c_j = zd_j + ab_j$ are completely random and independent from each other, since the $d_j$'s are completely random and independent from each other (and the $b_j$'s). Hence, ignoring the negligible case of $z = 0$, we get precisely the right hand side of Equation (1).

Notice, the reduction above is *tight*, as we surprisingly did not use the hybrdi argument! □

We remark that SKE constructions above, namely those stateful schemes that simply keep outputting a steam of pseudorandom bits (to be used as one-time pads), are called *stream ciphers*. They should be contrasted with *block ciphers* we will mention the next lecture.

## 2 Criticism + Looking Ahead

We notice that in stateful schemes, the sender and the receiver must be synchronized at all times to ensure correctness of decryption. This may be achieved by either agreeing on the policy of updating the secret key at encryption and decryption of each message. This is a disadvantage.

As one was to circumvent this, assume we can ensure that the state of the scheme has the form $(s, count)$, where $s$ is the "real" secret key, which *never changes*, while *count* is a simple counter that tells how many messages have been encrypted so far. Namely, the only change to the state is the instruction $count = count + 1$. In this case, even if the synchronization is temporarily lost, the sender and the recipient can exchange their counters, and reset the counter to the maximum value. It is easy to see that this exchange of counters will not conflict with IND-security, while will partially eliminate the problem of synchronization.

We notice, however, that our scheme from the previous section is not of this form. Thus, the first question we ask is:

- **Question 1:** Can we build a deterministic stateful CPA-secure SKE with the counter, as explained above?

The second question is whether we can have a stateless SKE, as we had for PKE. Naturally, this has to be randomized.

- **Question 2:** Can we build a randomized stateless CPA-secure SKE?

Finally, assuming the answer to the questions above is "yes",

- **Question 3:** What achieves greater security/efficiency: (state+determinism) or (no state+randomization)?

To answer all these questions, we introduce the concept of Pseudo Random Function Family (PRF), a really strong cryptographic primitive with several interesting properties. Afterwards, we present a very important application of PRFs, namely a new argument technique which allows us to separate the discussion of efficiency issues from the analysis of the security of the system.

# 3 Pseudo Random Function Family

## 3.1 Introduction

When we introduced Pseudo Random Generators, we saw that they are "efficient stretchers of randomness": given a truly random $k$-bit string, a PRG outputs a much longer (but polynomial in $k$) stream of bits that "looks random" with respect to any real (i.e. PPT) algorithm. Now we want to go further, and try to answer the question: can we do better? In other words, can we extract an exponential amount of pseudo random bits from a $k$-bit long seed?

Stated this way, this question does not really make sense, since it is impossible to even write down such a sequence efficiently. Anyway, we may want to know whether we can get an *implicit* representation of an exponential number of bits. One well-known class of exponentially long objects having "short" descriptions is that of *computable functions*, since the dimension of a mapping from $\{0,1\}^{\ell}$ to $\{0,1\}^{L}$ is $2^{\ell} \cdot L$.
After all, our question may be rephrased as:

> Can we use a $k$-bit long seed $s$ to efficiently sample a computable function $f_s$ from the space $\mathcal{R}(\ell(k), L(k))$ of all possible functions $F : \{0,1\}^{\ell(k)} \rightarrow \{0,1\}^{L(k)}$, in an 'almost-random' manner?

## 3.2 Definition

The essence of the above question can be formalized by the following definition.

Definition 2 [Pseudo Random Function Family]
A family $\mathcal{F} = \langle f_s \mid s \in \{0,1\}^k \rangle_{k \in \mathbb{N}}$ is called a family of $(\ell(k), L(k))$ Pseudo Random Functions if:

- $\forall k \in \mathbb{N}, \forall s \in \{0,1\}^k, \quad f_s : \{0,1\}^{\ell(k)} \rightarrow \{0,1\}^{L(k)}$;
- $\forall k \in \mathbb{N}, \forall s \in \{0,1\}^k, \quad f_s$ is polynomial time computable;
- $\mathcal{F}$ is pseudo random: $\forall$ PPT Adv

$$\left| Pr[\mathsf{Adv}^{f_s}(1^k) = 1 \mid s \leftarrow_R \{0,1\}^k] - Pr[\mathsf{Adv}^{F}(1^k) = 1 \mid F \leftarrow_R \mathcal{R}(\ell(k), L(k))] \right| \leq \mathsf{negl}(k)$$

$\diamondsuit$

In other words, for a family $\mathcal{F}$ to be pseudo random, it is not required for its generic element $f_s$ to be indistinguishable from a function $F$ drawn at random from $\mathcal{R}(\ell(k), L(k))$; rather, the *behavior* of any PPT adversary Adv which is given oracle access to the function $f_s$ must be indistinguishable from the behavior of the same adversary when given oracle access to a function $F$:

$$\forall \text{ PPT Adv} \qquad \text{Adv}^{f_s} \approx \text{Adv}^{F}$$

To put this yet in another way, imagine there are two distinct worlds: in the first world the adversary Adv queries a function chosen from the family $\mathcal{F}$, while in the second world the adversary's queries are answered by a truly random function $F$. Here by "truly random function" we mean a black box which outputs a fresh random value on each invocation, except that it is *consistent*, i.e. if queried twice on the same value, it always returns the same output. Now we say that $\mathcal{F}$ is a good family of PRFs if, although the outputs given by $f_s$ are clearly correlated while $F$'s answers are completely independent, the behavior of the adversary is essentially the same, so that it is not possible to tell these two worlds apart.

**Comments.**
Observe that this is a very strong requirement: how is it possible that no efficient adversary can realize whether it has been interacting with a function $f_s$ that uses only $k$ bits of randomness or with a function $F$ that consist of $2^{\ell(k)} \cdot L(k)$ random bits? A partial answer is that the adversary can only make polynomially many queries, and thus it doesn't have enough time to infer which "world" it is in.

### 3.3 PRFs vs PRGs

Our initial intent was to generalize the notion of PRG: this is indeed the case, since it actually turns out that PRGs can be viewed as a particular instantiation of PRFs.

In the case of a PRF, the adversary is given oracle access to the chosen function $f_s$; in the case of a PRG, since the output of a generator $G$ is just polynomially long (say $k^c$), the adversary can be given the entire string $G(s)$.

Anyway, it is trivial to simulate the knowledge of the string $G(s)$ using oracle access to a function $f_s$: the adversary may ask which bit it wants to know (specifying its position), and the oracle replies with the value of that bit. Since the position of one bit in a $k^c$ long string can be determined using $c \log k$ bits, and the answer is always one bit long, PRGs may be thought as PRFs where $\ell(k) = c \log k$ and $L(k) = 1$.

Therefore, for $\ell(k) = O(\log k)$, PRFs *degenerate* to PRGs; to ensure a gain in power with respect to PRGs, we have to enforce the *non-triviality* condition: $\ell(k) = \omega(\log k)$.

Notice, the moment non-triviality of the *input* length is ensured, there is no theoretical reason to lower bound for the *output* length value $L(k)$. This is because since any "non-trivial" family $\mathcal{F}$ of PRFs, — even the one with output lengh $L(k) = 1$, — can be easily extended to a family $\mathcal{F}'$ with $L(k) = k^c$: for each $f_s \in \mathcal{F}$, we include in $\mathcal{F}'$ the function

$f'_s : \{0,1\}^{\ell(k)-c\log k} \to \{0,1\}^{k^c}$ defined as follows:

$$f'_s(x') = \underbrace{f_s(\overbrace{0\ldots0}^{c\log k}\, x') \circ f_s(\overbrace{0\ldots1}^{c\log k}\, x') \circ \ldots \circ f_s(\overbrace{1\ldots1}^{c\log k}\, x')}_{k^c \text{ bits}}$$

Of course, in practice evaluating such a function bit-by-bit is very slow, but it demostrates that worrying about the output length is somewhat of a secondary issue, as long as we can construct a PRF with a "non-trivial" input length.

# 4 A general construction for PRFs

Once we have defined the notion of PRF family, we look at the problem of building such a family, and of the minimal assumption for the construction to go through. Surprisingly, it turns out that the existence of PRG implies the existence of PRF, although the transformation is too elaborated to be useful in practice. This result is due to Goldreich, Goldwasser and Micali, and is therefore known as **GGM construction**.

The GGM construction presents a loose resemblance to the technique used to obtain an IND-CPA secure stateful SKE scheme from any length-doubling PRG $G$. Recall from the previous lecture that to this aim we denote by $G_0(x)$ and $G_1(x)$ respectively the first and the second halves of the output of $G(x)$:

$$G : \{0,1\}^k \to \{0,1\}^{2k} \qquad G_0, G_1 : \{0,1\}^k \to \{0,1\}^k$$
$$G(x) \doteq G_0(x) \circ G_1(x)$$

To encrypt the first message, we apply $G$ to the shared key $s_0$ and set the new state $s_1$ to be $G_0(s_0)$, while masquerading the message with the pad $p_1 = G_1(s_0)$. The next time a message must be encrypted, the generator $G$ will be applied to the new state $s_1$, yielding $s_2 = G_0(s_1)$ and $p_2 = G_1(s_1)$. We can think of the whole process as the construction of an *unbalanced* binary tree (sketched in figure 1), in which we always go down to the left: the problem with this approach is that to have $n$ leaves, we have to construct a tree with height $n$.

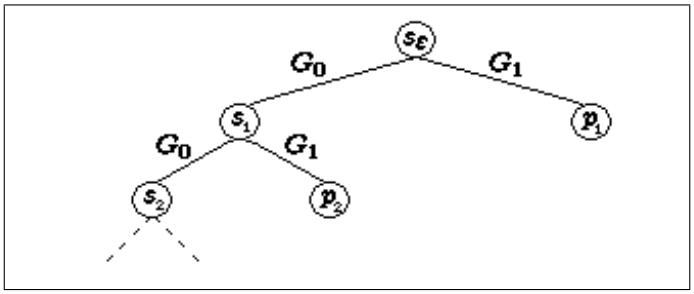



Figure 1: The unbalanced binary tree in the stateful SKE construction.

Clearly, it would be much more efficient to construct a *complete* binary tree, since this would give $2^n$ leaves on a tree of height $n$. To do so, for each fixed $s \in \{0,1\}^k$ we define

$G_x(s)$ through the following recursion:

$$G_\varepsilon(s) = s$$
$$G_{0\circ\bar{x}}(s) = G_{\bar{x}}(G_0(s))$$
$$G_{1\circ\bar{x}}(s) = G_{\bar{x}}(G_1(s))$$

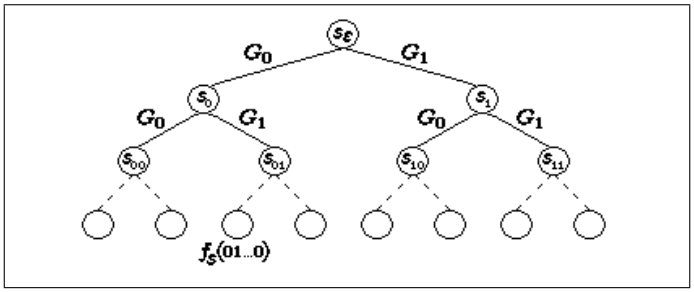



Figure 2: The complete binary tree in the GGM construction.

But how does this construction relate to Pseudo Random Functions? To sample a function $f_s : \{0,1\}^{\ell(k)} \to \{0,1\}^k$ given a random seed $s \in \{0,1\}^k$, define $f_s(x) = G_x(s)$. In this way, each input $x$ identifies a path in the complete binary tree having $s$ as root, and the output of the function $f_s$ is the value associated to the leaf down such path. According to the definition of $G_x(s)$, to compute the function $f_s$ on a single input it is necessary to evaluate the generator $G$ $\ell(k)$ times, which is still polynomial, although not very efficient.

It is not obvious that the above construction defines a family $\mathcal{F}$ of PRFs: we are extracting $2^{\ell(k)}$ different values out of one single truly random string! Still, no efficient adversary bears a (significantly) different behavior whether it interacts with this function or with a genuine random function (i.e. a function in which the $2^{\ell(k)}$ "leaves" are all random values). This claim is proved in the following theorem, which makes an extensive use of the hybrid argument.

**Theorem 1 (Goldreich-Goldwasser-Micali)**
*The family $\mathcal{F} = \langle f_s \mid s \in \{0,1\}^k \rangle_{k\in\mathbb{N}}$ where $f_s(x) = G_x(s)$ (as explained above) is a family of $(\ell(k), k)$ Pseudo Random Functions.*

**Proof:** To prove this theorem we want to use the hybrid argument: anyway, in this case the situation is a little different from what we have seen so far, since what we want to prove is not that a single pair of objects are computationally indistinguishable (like in $G(s) \approx R$), but rather that an infinity of related pair of objects are indistinguishable from each other:

$$\forall \text{ PPT Adv} \qquad \mathsf{Adv}^{f_s} \approx \mathsf{Adv}^{F}$$

Accordingly, to apply the hybrid argument, we need to find a sequence of *oracles* $T_0 \ldots T_{\mathsf{poly}(k)}$ such that $f_s \equiv T_0, F \equiv T_{\mathsf{poly}(k)}$, and for all the intermediate oracle it holds that:

$$\forall \text{ PPT Adv} \qquad \mathsf{Adv}^{T_i} \approx \mathsf{Adv}^{T_{i+1}}$$

Let's start defining the appropriate sequence of oracles. Observe that both $f_s$ and $F$ are queried from the adversary $\mathsf{Adv}$ about the value (in some specific point) of the function they "represent". As a consequence, we can think of those oracles as a set of $2^{\ell(k)}$ nodes, each containing the value of the function in one of the possible inputs.

In the case of $f_s$, this nodes can in turn be thought as the leaves of a complete binary tree of height $\ell(k)$, whose root contains the seed $s$: this is the tree we talked about when discussing the construction of the function $f_s$ from the $\mathsf{PRG}$ $G(x) = G_0(x) \circ G_1(x)$. We can think in a similar way also about the oracle $F$, even if in the case the "tree structure" is not inherent to its construction: all the nodes in the tree are "empty" nodes, except for the leaves, which contain all the random values of the $F$.

Stated this way, it is easier to figure out a possible way to "smoothly change" the oracle $f_s$ into the oracle $F$: instead of having randomness only in the root, and computing all the rest of the tree (and in particularly the leaves, using the $\mathsf{PRG}$ $G$ (as in $f_s$); or having all the randomness in the leaves, so that nothing is to be computed pseudorandomly (as in $F$), we can define the intermediate oracle $T_i$ to have an empty tree structure from level 0 to level $i-1$, all nodes at level $i$ containing truly random values, and the rest of the tree from level $i+1$ to level $\ell(k)$ (where we find the leaves, i.e. the values returned by this oracle) being calculated via successive applications of the pseudo random generator $G$.

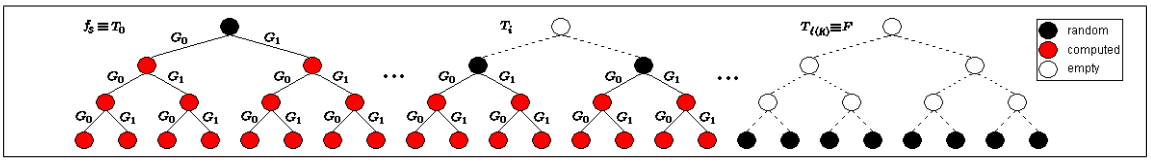



Figure 3: The sequence of oracles used in the first hybrid argument.

In this way we have constructed polynomially many intermediate oracles: in particular $f_s \equiv T_0$, since the randomness is only at level 0 (i.e. the root), and all the rest is computed through applications of $G$; in addition, $F = T_{\ell(k)}$, since all the structure above the level $\ell(k)$ (i.e. the last level of the tree), is empty, and the randomness is contained directly in the leaves (see figure 3). Therefore, by the hybrid argument, we can reduce the proof of the theorem to proving that:

$$\forall\ \mathsf{PPT}\ \mathsf{Adv} \qquad \mathsf{Adv}^{T_i} \approx \mathsf{Adv}^{T_{i+1}}$$

To this aim, let us fix an arbitrary adversary $\mathsf{Adv}$, and prove that $\mathsf{Adv}^{T_i} \approx \mathsf{Adv}^{T_{i+1}}$: having shown that, from the generality of such adversary the above statement will hold true, and thus the proof of this theorem will follow.

Since $\mathsf{Adv}$ is a $\mathsf{PPT}$ algorithm, it can do at most a polynomial number of queries to its oracle, say $t = \mathsf{poly}(k)$. In order to prove that the behavior of $\mathsf{Adv}$ with oracle $T_i$ is indistinguishable from the behavior of $\mathsf{Adv}$ with oracle $T_{i+1}$, we want to use again the hybrid argument: let's look at the correct sequence of intermediate oracle to use.

It would be tempting to consider the sequence of oracles $\overline{T_{ij}}$ in which the randomness is contained in the first $j$ nodes at level $i$, and in the children (at level $i+1$) of the remaining nodes at level $i$ (see figure 4).

Clearly the two extreme of such sequence would be $T_i$ and $T_{i+1}$, but how many intermediate oracles would result? At level $i$ there are $2^i$ nodes, and so when $i$ approaches $\ell(k)$ there

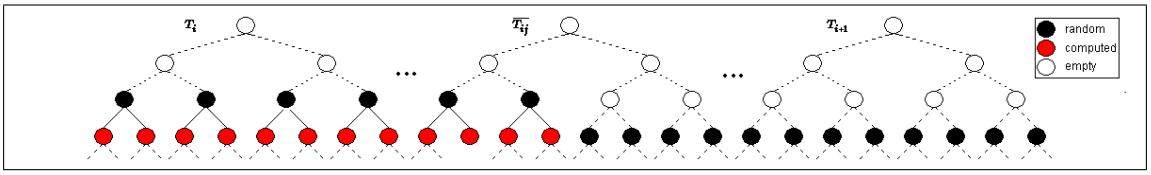



Figure 4: A first (wrong) step towards the second hybrid argument.

would be exponentially many intermediate oracles, way too much for the hybrid argument to go through.

How can we find a way out this situation? Recall that the adversary $\mathsf{Adv}$ queries its oracle at most $t$ time; intuitively, the problem with the intermediate oracles $\overline{T_{ij}}$ was that they induced a "too fine" differentiation: since $\mathsf{Adv}$ makes just $t$ queries, having $t$ intermediate oracles must suffice.

Accordingly, for $j \in [0..t]$, define the intermediate oracle $T_{ij}$ as follows: to answer each of the first $j$ queries posed by $\mathsf{Adv}$, the oracle $T_{ij}$ associates a random value to the unique node at level $i$ that lies in the path from the root to the leaf containing the value requested by the query, and then it computes the requested value via $\ell(k) - i$ applications of the generator $G$ (notice that up to now this is exactly the behavior born by the oracle $T_i$). Beginning with the $(j+1)^{st}$ query, the oracle $T_{ij}$ starts behaving like $T_j$: to respond to a request for the value associated with a certain leaf, the oracle picks a random value and puts it inside the ancestor at level $i+1$ of the leaf at hand; afterwards, it computes (as usual) the desired value through $\ell(k) - i - 1$ calls to the $\mathsf{PRG}$ $G$. There is a last technicality to be added to completely specify the oracle $T_{ij}$: it acts consistently, i.e. if, while looking at the path from the root to the leaf associated to the value requested by $\mathsf{Adv}$, the oracle $T_{ij}$ finds out that an ancestor of that leaf has already been filed out (in answering a previous query), it uses that ancestor to compute the value of the leaf, without adding any new randomness to the tree.

Now this sequence is well-suited: it consists of $t+1$ intermediate oracles, and $T_i \equiv T_{i0}$, while $T_{i+1} \equiv T_{it}$. To complete this hybrid argument, it is left to prove that $\mathsf{Adv}^{T_{ij}} \approx \mathsf{Adv}^{T_{ij+1}}$. But this is of course the case, since the only difference between the two oracles is that one answered $\mathsf{Adv}$'s $(j+1)^{st}$ query putting a random value $z$ at level $i$ (and thus filling its left child $l$ and its right child $r$ with $G_0(z)$ and $G_1(z)$ respectively), while the other answered the same query putting two random values $R_1$ and $R_2$ respectively in $l$ and $r$. If $\mathsf{Adv}$ behaves differently in the two cases, it would imply that $\mathsf{Adv}$ is able to distinguish between $G(z) \doteq G_0(z) \circ G_1(z)$ and $R_1 \circ R_2 \equiv R$, or, in other words, $G(z) \not\approx R$, contradicting the pseudorandomness of the generator $G$ used in the GGM construction. It follows that $\mathsf{Adv}^{T_{ij}} \approx \mathsf{Adv}^{T_{ij+1}}$, for any $j$, which entails (by the hybrid argument) that $\mathsf{Adv}^{T_i} \approx \mathsf{Adv}^{T_{i+1}}$. From the arbitrariness of $\mathsf{Adv}$, this holds true for any $i$, and finally (again by the hybrid argument):

$$\forall\ \mathsf{PPT}\ \mathsf{Adv} \qquad \mathsf{Adv}^{f_s} \approx \mathsf{Adv}^{F}$$

□

**Remark 1** *This is by far the most intensive use of the hybrid argument we have seen: it is actually so intense that in the reduction we lose an important fraction of the advantage.*

*This is because, from the proof of the validity of the hybrid argument (see Lecture 5), we know that if the advantage in distinguishing two intermediate distributions is $\epsilon$, then the advantage in distinguishing between the two initial distributions can increase by a factor equal to the number of intermediate elements. Therefore, in our case, we are losing a factor of $t \cdot \ell(k)$ in comparison with the advantage in breaking the initial* PRG.

# 5 More Efficient Construction under DDH

The GGM construction is very interesting: it features an original application of the hybrid argument and it demonstrates the use of complete binary trees to build complex primitives out of known, more basic ones. Anyway, the structure of the reduction is such that the construction loses both in efficiency and in security with respect to the underling "building block", namely the pseudo random generator $G$.

For these reasons, PRFs to be used in practice cannot be obtained in this way: a more concrete, number-theoretic construction is needed. Below we present one of the best-to-date practical (and yet provable!) construction, due to Naor-Reingold, which is based on the DDH assumption.

The construction works in the group $\mathbb{G} = QR_p$ of quadratic residues modulo $p$, where $p$ is a strong prime (i.e. it is of the form $p = 2q + 1$, for some prime $q$.) In this setting, the DDH assumption can be stated as:

$$\langle g, g^a, g^b, g^{ab} \rangle \approx \langle g, g^a, g^b, g^c \rangle$$

where $g$ is a random generator of $\mathbb{G}$ and $a, b, c$ are chosen uniformly at random in $\mathbb{Z}_q$.

For a given choice of the (public) parameters $p, q, g$, the Naor-Reingold pseudo random function family is $\mathcal{NR} = \langle NR_{p,g,a_0,a_1,\ldots,a_\ell} \mid a_0, a_1, \ldots, a_k \leftarrow_R \mathbb{Z}_q \rangle_{\ell \in \mathbb{N}}$, where each function $NR_{p,g,a_0,a_1,\ldots,a_\ell} : \{0,1\}^\ell \to \mathbb{G}$ is defined as follows:[1]

$$NR_{p,g,a_0,a_1,\ldots,a_\ell}(x_1, \ldots, x_\ell) = (g^{a_0})^{\prod_{i \in S(x)} a_i} \qquad \text{where } S(x) = \{i \geq 1 \mid x_i = 1\}$$

In other words, the input $x = x_1, \ldots, x_\ell$ is considered bit by bit and, for each $x_i$ equal to 1, the corresponding value $a_i$ is included in the modular exponentiation. The advantage of such definition is that the value of the function on a particular point can be computed with $O(\ell)$ multiplications: we can compute the exponent $\alpha = a_0 \prod_{i \in S(x)} a_i \bmod q$ with at most $\ell$ multiplications, and then compute $g^\alpha \bmod p$ with at most $2\ell$ multiplications (using the "Square & Multiply" algorithm.)

The *NR* construction for PRFs can be though as a particular instantiation of the complete binary tree technique seen in the GGM construction, where the PRG used is $G_{p,q,g,a}(g^b) \doteq G'_0(g^b) \circ G'_1(g^b) = \langle g^b, g^{ab} \rangle$. However, in some sense, the solution proposed by Naor Reingold also generalizes the previous construction, since a different exponent $a_i$ is considered at each level of the tree, while in the standard GGM the same PRG is used at all levels. This is shown in figure 5.

[1] Notice, the output of the construction is a random element of $\mathbb{G}$. However, we already know a deterministic map that can turn it into a random element of $\mathbb{Z}_q$.

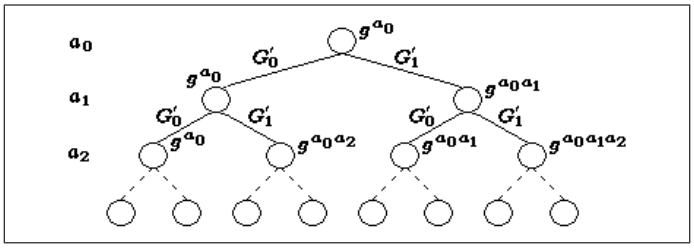



Figure 5: The complete binary tree in the NR construction.

The proof of security of this PRF family bears a close resemblance to the proof of the previous theorem; consequently, we state the result without explicitly including the supportive argument. We mention, however, that the actual security of the NR PRF is better that that of the GGM construction. In particular, the former "lost" a factor $t\ell$ in its security, where $t$ is the qumber of PRF queries made by the attacker and $\ell$ is the input length. In contrast, a clever "random self-reducibility" argument from Lemma 1 allowed Naor and Reingold to lose "only" a factor $\ell$ in the security reduction. Namely, we only use the "outer" $\ell$ hybrids of the GGM construction (one per level of the tree). Inside each level $i$, we use Equation (1) (with value $a = a_i$) to directly show that giving oracle access to $T_i$ is indistinguishable from giving oracle access to $T_{i+1}$, under the DDH assumption.

**Theorem 2 (Naor Reingold)**
*Under the* DDH *assumption, the family* $\mathcal{NR} = \langle NR_{p,g,a_0,a_1,\ldots,a_\ell} \mid a_0, a_1, \ldots, a_\ell \leftarrow_R \mathbb{Z}_q \rangle_{\ell \in \mathbb{N}}$ *is a* PRF *family with* $O(\ell)$ *security loss when reduced to DDH.*

| CSCI-GA.3210-001<br>MATH-GA.2170-001 | **Introduction to Cryptography** | Macrh 21, 2012 |
|---|---|---|

# Lecture 9

*Lecturer: Yevgeniy Dodis* *Spring 2012*

Last time we introduced the concept of Pseudo Random Function Family (PRF), and showed a generic construction of PRFs from PRGs, as well as a more efficient construction under the DDH assumption. This lecture we present several important applications of PRFs. First, a new argument technique which allows us to separate the discussion of efficiency issues from the analysis of the security of the system. And, second, how to apply PRFs to easily build CPA-secure symmetric cryptosystems. In particular, we will answer the 3 questions we asked in the previous lecture.

## 1 Applications of PRFs

Pseudo Random Functions are a very powerful cryptographic tool: their key property — no efficient algorithm substantially changes its behavior whether it interacts with a pseudo random function or a truly random one — is so strong that we can do a lot of things with PRFs. Let's look at some simple applications.

### 1.1 Identify Friend or Foe

The problem of identifying friend or foe (IFF) consists in deciding, in a dynamic setting, whether you are facing an enemy or an ally. Consider an air battle between two parties $A$ and $B$, in which all the planes belonging to the same air force share a secret value, and let $i$ and $j$ be respectively the secret associated with parties $A$ and $B$. Before shooting a potential target, a warplane of party $A$ challenge the target with a random $r$: if the target reply with $f_i(r)$, then it is identified as a friend and it is not destroyed. In this scenario, the adversary will not be able to reply correctly, since even after seeing many pairs $(r', f_i(r'))$, he still has negligible probability in predicting the value $f_i(r)$ for an unseen, random $r$. Concretely, if the attacker saw up to $q$ previous $(r', f_i(r'))$ pairs, and (negligibly small) $\varepsilon$ is the security of our PRF against a distingusiher making at most $q$ PRF queries, then the maximum probability the attacker can predict the $k$-bit value $f_i(r)$ is at most $\varepsilon + \frac{q}{2^k}$, which is negligible. Moreover, since this setting is dynamic, the adversary has no time to mount a "man-in-the-middle" attack, forwarding the challenge to another warplane: indeed, the adversary itself cannot distinguish its friends from its enemies!

### 1.2 The Random Function Model

The most important application of Pseudo Random Functions is that they enable a higher level technique to argue about security. Given a cryptographic scheme which uses PRFs, to prove its security against an adversary, we consider the chances the adversary Adv has to break the system in the *Random Function Model*, where *every pseudo random function* $f_s$

*is replaced with a truly random function $F$.* What we gain from this transformation is that it is usually simpler to deal with security in the Random Function Model, since one can make use of Information-Theoretic considerations to show that the adversary's advantage (whatever it is defined to be) is negligible. Then, we can argue that Adv's advantage remains negligible even in the original scheme, because otherwise we would have found an efficient algorithm (the adversary Adv) whose behavior is significantly different whether it interacts with a PRF or a truly random one, contradicting the definition of PRF.

In brief, to argue about a cryptographic scheme that uses PRFs, we proceed as follows:

1. we discuss its *efficiency* in the PRF world;
2. we prove the *security* (information-theoretically) in the Random Function Model;
3. we conclude that the original system is also secure when using PRF, since otherwise we would contradict the pseudo randomness of the PRF family.

The only thing to care of is that in going from point 1 to point 2, it is allowed to substitute random functions only in place of pseudo random function $f_s$ whose seed $s$ is never shown to the adversary Adv, because otherwise $f_s$ will not "look random" to Adv.

This is a powerful technique, that allows us to quickly establish the security of complex constructions: we discuss two examples in the following subsections.

### 1.3 Construction of an IND-CPA-secure SKE with counter as state

Now that we have developed the new tool of PRFs, it is a simple matter to define a stateful SKE scheme IND-secure against chosen plaintext attack which uses a simple counter to maintain the state. This scheme is known as CTR *scheme.*

1. On input $1^k$, the *key-generation algorithm* $G$ chooses a family $\mathcal{F}$ of PRFs, and sets the public key $PK$ to be the description of that particular family (for example, in the case of the $NR$ construction, this consists of the choice of the strong prime $p = 2q+1$ along with a generator $g$ of $QR_p$.) Afterwards, $G$ takes a random seed $s \in \{0,1\}^k$ and sets $SK = s$. Finally, $G$ outputs $(PK, SK, M_k)$, where $M_k = \{0,1\}^{L(k)}$. The counter for both the encryption and decryption algorithm is initially set to 0.
2. To encrypt a message $m \in M_k$ when the counter is `ctr`, the *encryption algorithm* $E_{PK}$ outputs the ciphertext $c = f_s(\texttt{ctr}) \oplus m$, and increment its counter `ctr`.
3. In order to decrypt a ciphertext $c$, having counter `ctr`, the decryption algorithm $D$ compute $m = f_s(\texttt{ctr}) \oplus c$, and increment the value of `ctr`.

In attacking this cryptosystem, the adversary Eve knows the PRF family $\mathcal{F}$ being used, the security parameter $k$ and the current value of the counter (since she can easily keep track of all the ciphertexts Alice has sent to Bob.) Nevertheless, all that she knows can be expressed as the knowledge of polynomially many pairs of the form $(r', f_s(r') \oplus m')$ and this does not help her to gain any advantage in guessing $m$ from its encryption $f_s(\texttt{ctr}) \oplus m$.

It is possible to give a formal proof of the above claim using a standard reductionist argument; however, we prefer to use the higher level technique introduced in previous

section, both to demonstrate the use of such technique and because the argument becomes simpler.

**Theorem 1** *The* CTR *scheme defined above is an* IND-CPA*-secure* SKE.

**Proof:** The CTR scheme is clearly an SKE since both the encryption and the decryption algorithm can be efficiently computed (notice that $f_s$ is poly-time computable by the definition of PRF family), and the correctness property is trivially fulfilled.

Following the steps suggested in the previous subsection, we argue about the security of this scheme in the Random Function Model. Since the adversary does not know the seed $s$ used in the pseudo random function $f_s$, it is legitimate to replace this function with a truly random function $F$ within the Encryption and the Decryption algorithms:

$$\begin{array}{rll} E_F(m): & \textbf{set:} & c \leftarrow F(\texttt{ctr}) \oplus m, \quad \texttt{ctr} \leftarrow \texttt{ctr} + 1 \\ & \textbf{output:} & c \\ D_F(c): & \textbf{set:} & m \leftarrow F(\texttt{ctr}) \oplus c, \quad \texttt{ctr} \leftarrow \texttt{ctr} + 1 \\ & \textbf{output:} & m \end{array}$$

In the chosen plaintext attack, Eve, after having queried her encryption oracle a number of times, chooses a pair of messages $m_0$ and $m_1$ whose encryptions she believes to be able to distinguish from each other. Once challenged with the encryption $c = F(\texttt{ctr}) \oplus m_b$, for a random $b \in \{0, 1\}$, she can make polynomially more queries to the oracle, and then she has to decide which message was encrypted. How likely is she to succeed? We claim that her probability of success is exactly $1/2$.

To see why, notice that we are working in the Random Function Model, and thus the value $F(\texttt{ctr})$ is completely random and independent from the values $F(0), \ldots, F(\texttt{ctr} - 1)$ computed so far, and possibly known to Eve. Moreover, it is independent from all values used in subsequent queries made by Eve to her oracle, since the counter consists of $\ell(k)$ bits, and so it would take $2^{\ell(k)}$ queries for the counter to overflow and take again the same value. From the non-triviality condition for PRF families, this is more than polynomial, and thus Eve does not have enough time to wait until this happens. It follows that each time she ask the oracle to encrypt a message, she will get back a ciphertext $c = F(\texttt{ctr}') \oplus m'$ which is a completely random quantity (thanks to the randomness of $F(\texttt{ctr}')$), and thus she cannot learn anything from it. Therefore, the best she can do to guess the bit $b$ is flipping a coin, and hence the CTR scheme is IND-CPA-secure in the Random Function Model.

From the pseudorandomness of the family $\mathcal{F}$ we can now conclude that the original CTR scheme is IND-CPA-secure, since otherwise Eve would have a different behavior whether accessing the oracle $f_s$ or the random oracle $F$, contradicting the assumption that $\mathcal{F}$ is a family if PRFs. □

**Remark 1** *Notice that the advantage of Eve in the Random Function Model is exactly* $1/2$*, since although she has access to an oracle, she cannot learn anything from its responses, so that she cannot do anything better then guessing a bit at random. When we go from the Random Function Model back to the "real world", Eve gains at most the same negligible advantage possible in distinguish between the "two worlds", which is known to be at most negligible from the pseudorandomness of the family* $\mathcal{F}$*.*

## 1.4 Construction of a stateless IND-CPA-secure SKE

Once we have seen the CTR scheme, it is easy to modify it so that no state is required neither for encryption nor for decryption. Indeed, in the CTR SKE each ciphertext has the form $F(\texttt{ctr}) \oplus m$, and the legitimate recipient (Bob) is able to decrypt it because it maintains his state in sync with Alice's state. What if the synchronization is lost? The two parties can simply exchange their own states, since this would be of no help for the attacker.

But therefore, why do we need synchronization? Alice and Bob can simply exchange their states each time, or, even better, they can avoid keeping state at all, by having the encryption algorithm choose a new, fresh random value to be used as the "state" for the encryption at hand, and send it along with the "real" ciphertext to Bob. This leads to the definition of the XOR scheme, included below.

1. On input $1^k$, the *key-generation algorithm* $G$ chooses a family $\mathcal{F}$ of PRFs, and sets the public key $PK$ to be the description of that particular family (for example, in the case of the *NR* construction, this consists of the choice of the strong prime $p = 2q + 1$ along with a generator $g$ of $QR_p$.) Afterwards, $G$ takes a random seed $s \in \{0,1\}^k$ and sets $SK = s$. Finally, $G$ outputs $(PK, SK, M_k)$, where $M_k = \{0,1\}^{L(k)}$.

2. To encrypt a message $m \in M_k$, the *encryption algorithm* $E_{PK}$ chooses a random value $r \in \{0,1\}^k$, compute $c = f_s(r) \oplus m$ and outputs the ciphertext $c' = \langle r, c \rangle$.

3. In order to decrypt a ciphertext $c' = \langle r, c \rangle$, the decryption algorithm $D$ computes the "pad" $f_s(r)$ and then the message $m = f_s(r) \oplus c$.

**Theorem 2** *The* XOR *scheme defined above is an* IND-CPA*-secure* SKE*.*

**Proof:** The XOR scheme is clearly an SKE since both the encryption and the decryption algorithm can be efficiently computed; in addition, since the value used to generate the pad is sent to the intended recipient of the message, the correctness property is also satisfied.

More interesting is the discussion of security: again, we consider the resilience of the XOR scheme from a chosen plaintext attack, in the Random Function Model. Since the adversary does not know the seed $s$ used in the pseudo random function $f_s$, it is legitimate to replace this function with a truly random function $F$ within the Encryption and the Decryption algorithms, obtaining:

$$
\begin{array}{rll}
E_F(m): & \textbf{set:} & r \leftarrow_R \{0,1\}^{\ell(k)}, \quad c \leftarrow F(r) \oplus m \\
 & \textbf{output:} & c' = \langle r, c \rangle \\
D_F(c): & \textbf{set:} & m \leftarrow F(r) \oplus c \\
 & \textbf{output:} & m
\end{array}
$$

Suppose that Eve has queried the encryption oracle a total of $q = q(k)$ times. Let $r_1 \ldots r_q$ be the random values chosen by the encryption oracle. And let $r$ be the random value used to encrypt the challenge ciphertext $\langle r, F(r) \oplus m_b \rangle$, where bit $b$ is random. It is clear the only way the attacker gets any information about the bit $b$ is if the "challenge" value $r$ belongs to the encryption oracle choices $\{r_1 \ldots r_q\}$. But since $r$ is random, this

probability is at most $q/2^\ell$, which is negligible. Therefore, in the Random Function Model the probability of success is at most $1/2 + q/2^\ell = 1/2 + \mathsf{negl}(k)$. Hence the XOR scheme is IND-CPA-secure in the Random Function Model; from the pseudorandomness of the family $\mathcal{F}$ used in actual XOR construction, we can finally conclude that the original XOR scheme is IND-CPA-secure. □

**Remark 2** *It is worth pointing out that in th case of the* XOR *scheme Eve has an advantage slightly better than guessing even in the Random Function Model. Although this make no difference in an asymptotic sense (since her advantage is anyway negligible), in practice this can be an issue, since it may leads to the necessity of using bigger values for the security parameter $k$, thus worsening, in the long run, the efficiency performance of the system.*

*This consideration shows that there is no simple answer to the third question we asked about* SKE *schemes; namely whether it is better to have stateful or randomized encryption schemes. Both the two approaches have their* pro's *and* con's*: the* CTR *scheme is a little bit inconvenient due to the necessity of maintaining a state, while the* XOR *scheme, albeit stateless, may look unsatisfactory in terms of exact security, and also because the length of the ciphertext is clisghly shorter (because it has to include the value $r$).*

### 1.5 Encrypting Long Messages

As we will see shortly, "practical" PRFs typically have fixed input and outputr size, often roughly equal to the security parameter $k$ (say, both being 128 bits). How can we use such PRFs to encrypt arbitrarily long messages? There are several ways out.

First, we notice that the CTR and XOR scheme only require long outputs size to support long messages. And we already mentioned that we can expand the output by a factor $2^a$ by "wasting" $a$ input bits. Thus, to encrypt 1 Gigabyte $\approx 2^{30}$ file using $L = \ell = 128 = 2^7$ one can create a new PRF with input size $128 - (30 - 7) = 103$ and output size $2^{30}$. Indeed, we will later study this scheme (under a different, but equivalent definition) which will be called "CTR mode of operation".

However, we will see that there are several other popular and effective ways to go about supporting long messages from "fixed length" PRFs. Historically, however, these methods, which will be called *modes of operation* originated from a stronger cryptographic primitive called a *block cipher*, or a *pseudorandom permutation* (PRP).[1] As it turned out, some of these modes of operations, like the counter mode, indeed work even better using regular PRFs. Others, like the "CBC mode" that we study shortly actually require the stronger structure of PRPs. Therefore, we make a brief detour and study PRPs, and then come back to various modes of operations used to encrypt long messages.

## 2 Pseudo Random Permutation Family

The concept of *Pseudo Random Permutations* (PRPs) is motivated by the desire to create a length-preserving encryption, so that we do not have to transmit a lot of additional bits for each bit of data that we encrypt. By now we know that such an encryption cannot be

[1] Usually, the term "block cipher" is used to describe a "fixed length PRP".

semantically secure, but the introduction of PRPs happens to be nevertheless useful both from practical and theoretical perspectives. We now make several definitions.

## 2.1 Definitions

We start with the following definition of a PRP:

DEFINITION 1 [Pseudo Random Permutation Family] A family $\mathcal{G} = \langle g_s : \{0,1\}^{\ell(k)} \rightarrow \{0,1\}^{\ell(k)} \mid s \in \{0,1\}^k \rangle_{k \in \mathbb{N}}$ is called a family of Pseudo Random Permutations if the following properties hold:

1. $g_s$ is a permutation, $\forall s$.
2. $g_s$ and $g_s^{-1}$ are both efficiently computable, $\forall s$.
3. $g_s$ is pseudo random: $\forall$ PPT Adv

   $$\left| Pr[\ \mathsf{Adv}^{g_s}(1^k) = 1 \mid s \leftarrow_R \{0,1\}^k\ ] - Pr[\ \mathsf{Adv}^{P}(1^k) = 1 \mid P \leftarrow_R \mathcal{P}(\ell(k))\ ] \right| \leq \mathsf{negl}(k)$$

   where $P$ represents a random *permutation* on $\ell(k)$ bits, chosen from the set of all such permutations $\mathcal{P}(\ell(k))$.

$\diamondsuit$

This definition is very similar to the definition of PRF from the previous lecture, and it would be a good idea to look back and compare. Again, when we state that $g_s$ is pseudo random, we mean that if Adv is given oracle access to $g_s$ (in particular, he does not know $s$) he cannot tell it apart from access to a "fake" oracle $P$ which represents a *truly random* permutation. Using indistinguishability notation:

$$\forall \text{ PPT Adv} \qquad \mathsf{Adv}^{g_s} \approx \mathsf{Adv}^{P}$$

Again, be sure to refresh your memory of PRFs from the definition in the previous lecture notes, and compare it with our definition of a PRP.

A stronger, and often required form of PRP is that of a strong PRP.

DEFINITION 2 [Strong PRP] A Strong PRP satisfies all the properties of a family of Pseudo Random Permutations, except the constraint [3.] is replaced by a stronger requirement below:

3'. $g_s$ is pseudo random even if oracle access to $g_s^{-1}$ is provided as well: for any PPT Adv,

$$\mathsf{Adv}^{(g_s,\ g_s^{-1})} \approx \mathsf{Adv}^{(P,\ P^{-1})}$$

$\diamondsuit$

This new definition provides us with an additional measure of security by allowing us to give Adv oracle access to $g_s^{-1}$.

### 2.2 Encryption from PRP's?

To partially motivate previous definitions, let us propose a naive "encryption" scheme that is length preserving: given a secret key $s$ and a message $m$,

$$c = E_s(m) = g_s(m) \qquad m = D_s(c) = g_s^{-1}(c)$$

If we let $g_s$ be taken from a family of Efficient PRPs, the scheme works, since $c$ can be decrypted it PPT. Also, since $E_s$ is a permutation, this encryption is length preserving, and we need not transmit any excess data as part of the ciphertext. However, there is a serious security flaw in the scheme: the adversary can tell if we send the same message twice, since the encryption is deterministic. Hence, the scheme is certainly not CPA-secure (even though it is obviously one-time, but this is uninteresting). Intuitively, though, the only thing that the adversary can learn from seeing several encryptions is which encryptions correspond to the same message. This is clearly the best we can hope for with length-preserving encryption: namely such encryption must look like a random permutation! However, can we build an efficient CPA-secure system from a PRP family?

As we will see, the answer is "yes". In fact, there are several ways to do it. However, we will examine these later, and start by relating PRFs and PRPs.

## 3 PRF $\Longleftrightarrow$ PRP

It turns out that any valid PRG is also a PRF, so if we wish to construct a PRF from a PRP, don't really have to do any work at all.

**Theorem 3** (PRP $\Rightarrow$ PRF) *Assume $\mathcal{G}$ is a* PRP *family. Then $\mathcal{G}$ is also a* PRF *family.*

Before turning to the proof of this statement, we need a result from Elementary Probability Theory, called the *Birthday Paradox*.

**Lemma 1 (Birthday Paradox)**
*Choosing at random $q$ values from a set of $N \gg q$ possible values, the probability of taking twice the same value is approximately $\frac{q^2}{2N}$.*

**Proof:** We have

$$\Pr[\text{"at least one repetition choosing } q \text{ times an element out of } N\text{"}] \leq$$
$$\leq \sum_{1 \leq i < j \leq q} \Pr[\text{"item } i \text{ collides with item } j\text{"}] = \binom{q}{2} \cdot \frac{1}{N} = \frac{q(q-1)}{2N} \leq \frac{q^2}{2N}$$

□

Now we can prove Theorem 3.

**Proof:** The proof uses the hybrid arguement. If we start by considering that members of $\mathcal{G}$ are indistinguishable from random permutations to the adversary Adv, all we need to show is that a random permutation is indistinguishable from a random function. In

other words, we must show $\mathsf{Adv}^P \approx \mathsf{Adv}^F$, where $P$ is a random permutation oracle and $F$ is a random function oracle. However, the only way the adversary can tell something is a random function and not a random permutation is if a collision occurs: namely, Adv gets the same output for two different inputs. But the Birthday Paradox discussed above tells us that the probability of such a collision is negligible as long as Adv can only make polynomially many calls to its oracle (which is true since Adv is PPT). Therefore, random permutations are indistinguishable from random functions, completing the proof. □

Next, we need to show the more difficult construction, PRPs from PRFs. In order to do this, we will take a brief detour to define something known as a Feistel Network.

### 3.1 Feistel Networks

The Feistel Network is a way of constructing an efficient, invertible permutation which can be neatly structured in "rounds". Feistel Networks generally consist of multiple rounds of a Feistel transformation, which we now define.

DEFINITION 3 [Feistel Transform] Let $f : \{0,1\}^k \to \{0,1\}^k$ be any efficient function. The *Feistel Transform* of $f$ is a permutation $H_f : \{0,1\}^{2k} \to \{0,1\}^{2k}$, given by: (below $L$ and $R$ are of length $k$ each)

$$H_f(L,R) = (R, \quad f(R) \oplus L)$$

We sometimes write $H_f(L,R) = (L',R')$ with $L' = R$ and $R' = f(R) \oplus L$ as a shorthand. ♢

In the following, we will always use $f$ which is a (pseudo)random function. In this case, the permutation $H_f$ essentially takes two inputs of $k$ bits, and outputs the right input in the clear (but on the left hand side of the output) and then uses the given function $f$ of the right input to "encrypt" the left input (the result of the encryption is on the right side of the output). Indeed, $(R, f(R) \oplus L)$ could be viewed as encryption of $L$ when $R$ is chosen at random and $f$ is a shared PRF. To summarize, the right side is somehow "mangled" and used to encrypt the left side, and then the two halves are swapped.

**Remark 3** *The Feistel Transform is indeed a permutation (even though $f$ need not be!), as can be seen by choosing a fixed $R$ first. It can be seen that the output $L' = R$ is the identity permutation of the right input $R$, and the output $R' = f(R) \oplus L$ for a fixed $R$ has become a permutation of the input $L$. Thus there is a unique output pair $(L',R')$ for each input pair $(L,R)$. More specifically, its inverse is given by:*

$$H_f^{-1}(L',R') = (f(L') \oplus R', \ L')$$

*This computation is strikingly similar to the original Feistel Transform, and in fact it is a "mirror image" of the Feistel Transform.*

Here are graphical representations for the definition of the Feistel transform and its inverse:

These picture suggest the possibility of "stacking" one Feistel Transform on top of another. In fact, the Feistel transform was originally designed to be efficient (computationally)

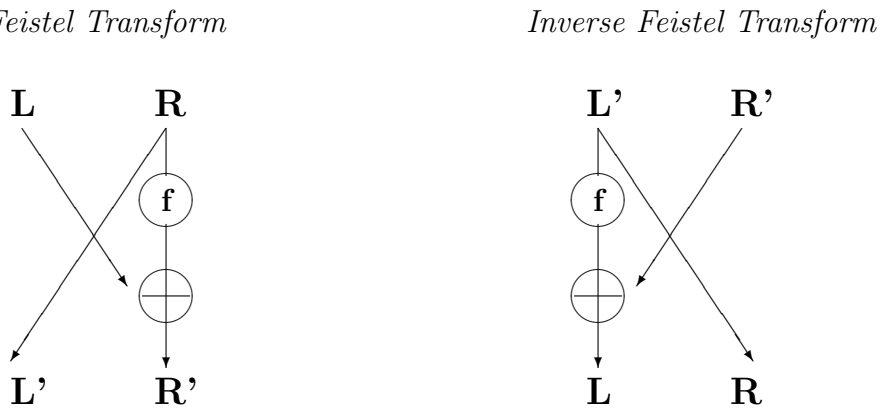



Figure 1: The Feistel Transform and Inverse Feistel Transform

when used in "multiple rounds", and this is where the power of the Feistel transform lies. Repeated applications of the Feistel transform constitute what is known as a Feistel Network. To visualize a Feistel Network, and the corresponding Inverse Feistel Network, one need only stack copies of the above drawings so that the outputs of one stage become the inputs to the next. There is one very important point that must be made: even though one can use the same $f$ from round to round, we will see that we should use *different* (pseudo)random $f$'s in different rounds. Of course, in this case to invert the Feistel Network, it is important to reverse the ordering of $f$'s as well.

DEFINITION 4 [Feistel Network] Given $f_1 \ldots f_n$, the corresponding Feistel Network, $H_{(f_1,f_2,\ldots,f_n)}$, is given by:

$$H_{(f_1,f_2,\ldots,f_n)} = H_{f_n}(H_{f_{n-1}}(\ldots(H_{f_1}(L,R))\ldots))$$

The operation performed by the Feistel Network is invertible, and the inverse may be obtained by the following (Inverse Feistel Network):

$$H^{-1}_{(f_1,f_2,\ldots,f_n)} = H^{-1}_{f_1}(\ldots(H^{-1}_{f_{n-1}}(H^{-1}_{f_n}(L',R'))\ldots)$$

$\diamondsuit$

These Feistel Networks are used heavily in real world cryptographic schemes (perhaps most notably, the *Data Encryption Standard* (DES)). It is difficult to prove anything about these Feistel Networks in their most general form, however, and the construction of many cryptographic schemes that use Feistel Networks is considered a "black art", since no one can really directly prove that it works. However, we *can* prove some *formal* results, by assuming that the functions $f_i$ used in the Feistel Network are independently drawn from a family $\mathcal{F}$ of PRFs! We now define a notation for such PRF Feistel Networks.

DEFINITION 5 [PRF Feistel Network] Let $\mathcal{F}$ be a PRF family whose functions map $k$ bits to $k$ bits (i.e. $\ell(k) = L(k) = k$). Let

$$\mathcal{H}^n_{\mathcal{F}} = \{H_{(f_1,f_2,\ldots,f_n)} \mid f_1, f_2, \ldots, f_n \in \mathcal{F}\}$$

We denote by $H^n_{\mathcal{F}}$ a random member of $\mathcal{H}^n_{\mathcal{F}}$, i.e. $H_{(f_1,f_2,\ldots,f_n)}$, where $f_1, f_2, \ldots, f_n$ are random *independently chosen* members of $\mathcal{F}$. That is, each round of the Feistel Transform uses a different, randomly chosen member of $\mathcal{F}$. $\diamondsuit$

Now that we have these definitions in hand, we can proceed to produce PRPs from PRFs by something known as the Luby-Rackoff Construction, which makes use of these PRF Feistel Networks.

### 3.2 The Luby-Rackoff Construction

**Theorem 4 (PRF $\Rightarrow$ (Efficient) PRP)** *Let $\mathcal{F}$ be a* PRF *family whose functions map $k$ bits to $k$ bits (i.e. $\ell(k) = L(k) = k$). Then $\mathcal{G} = \mathcal{H}^3_{\mathcal{F}}$ is an efficient* PRP *family. Namely,* 3 *rounds of the Feistel Network yield an efficient* PRP.

**Proof:** The proof is by hybrid arguement over a sequence of worlds, informally shown in the table below. The goal is to start with a World 0 in which we have only a PRF, and through a sequence of intermediate worlds that are each indistinguishable from the previous world, construct a World 5 that is just the desire random permutation on $2k$ bits.

| World | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Input | $(L, R)$ | $(L, R)$ | $(L, R)$ | $(L, R)$ | $(L, R)$ | $(L, R)$ |
| Inside | $f_1, f_2, f_3 \in \mathcal{F}$ | $f_1, f_2, f_3$ are truly random | $H_{f_1}(L, R)$ $\wr\!\!\wr$ $(R, \tilde{?})$ | $H_{f_1}(L, R)$ $\downarrow$ $H_{f_2}(R, \tilde{?})$ $\wr\!\!\wr$ $(\tilde{?}, \$)$ | $H_{f_1}(L, R)$ $\downarrow$ $H_{f_2}(R, \tilde{?})$ $\downarrow$ $H_{f_3}(\tilde{?}, \$)$ $\wr\!\!\wr$ | $P$ – random permutation |
| Output | $H_{f_1,f_2,f_3}(L, R)$ | $H_{f_1,f_2,f_3}(L, R)$ | $H_{f_2,f_3}(R, \tilde{?})$ | $H_{f_3}(\tilde{?}, \$)$ | $(\$, \$)$ | $P(L, R)$ |

Figure 2: Illustration of Luby-Rackoff Proof

The following *informal*, but informative, notation is used in the table. Do not worry if it is confusing for now. $(L, R)$ is used to represent a pair of inputs that are selected by the adversary Adv as the input to the oracle that he has. Clearly, many such pairs will be selected during the course of execution (wlog, we assume all the oracle calls are made with distinct inputs $(L, R)$). We use ? to represent some string is in under direct or indirect control of the adversary (in particular, the adversary might know part or all of it). We use $\tilde{?}$ to denote a value ? which is always distinct from one oracle call to another. Namely, the adversary has partial knowledge or control over selecting this value, but w.h.p. the adversary cannot cause a collision among different such values. Finally, we use \$ to denote a truly random $k$-bit string. Namely, from one oracle call to the other, this string is selected randomly and independently from previous calls.

As the first step, we look at the "real" World 0, where the oracle is $H_{f_1,f_2,f_3}$, where $f_1, f_2, f_3$ are PRF's. Recall from the previous lecture the discussion of "The Random Function Model". Using this model, we know that World 0 is indistinguishable from World 1,

where $f_1, f_2, f_3$ are truly random functions (technically, we have to use three hybrid arguments eliminating one $f_i$ every time, but this is simple). Jumping ahead, World 4 will be a truly random *function* from $2k$ to $2k$ bits. By Theorem 3, we knows that a truly random *function* is indistinguishable from a truly random *permutation*, i.e. World 4 is indistinguishable from World 5.

To summarize, the heart of the proof is to show that World 1 is indistinguishable from World 4: namely, $H_{(f_1,f_2,f_3)}$, where $f_1, f_2, f_3$ are random functions, is indistinguishable from a truly random function from $2k$ to $2k$ bits. Here is the intuition. We want to say that as the input $(L, R)$ travels through 3 layers of the Feistel network, the following changes happen:

$$(L,R) \quad \stackrel{H_{f_1}}{\longrightarrow} \quad (L_1,R_1)=(R,\tilde{?}) \quad \stackrel{H_{f_2}}{\longrightarrow} \quad (L_2,R_2)=(\tilde{?},\$) \quad \stackrel{H_{f_3}}{\longrightarrow} \quad (L_3,R_3)=(\$,\$)$$

Thus, the first layer only achieves that the right half $R_1$ never repeats from one oracle call to the other, but $\mathsf{Adv}$ may know a lot about both the left and the right parts. The second layer uses this to produce a string whose right half $R_2$ always looks random and independent from everything else, while the left half $L_2 = R_1$ may still be very predictable. Finally, the third layer uses the the randomness and unpredictability of $R_2$ to make both $L_3 = R_2$ and $R_3$ look totally random, as desired.

We now formalize the above intuition. World 2 is the same as World 1 with the following exception. If any two right halves $R_1$ happens to collide during the oracle calls, we stop the execution and tell $\mathsf{Adv}$ that a right-half collision happened. Thus, to show that $\mathsf{Adv}$ does not see the difference between World 1 and World 2, we must argue that the probability of collision is negligible. Assume $\mathsf{Adv}$ makes $t$ calls to the its oracle. Take any two such calls with inputs $(L, R) \neq (L', R')$. If $R = R'$, we must have $L \neq L'$, and hence

$$R_1 = L \oplus f_1(R) = L \oplus f_1(R') \neq L' \oplus f_1(R') = R'_1$$

i.e. the right half collision cannot happen then. On the other hand, assume $R \neq R'$. In order for $R_1 = R'_1$, we must have $L \oplus L' = f_1(R) \oplus f_1(R')$. Irrespective of the value $Z_\ell = L \oplus L'$, since $f_1$ is a truly random function and $R \neq R'$, $Z_r = f_1(R) \oplus f_1(R')$ is a truly random string, so the probability that $Z_\ell = Z_r$ is $1/2^k$. Hence, in both cases, the probability that $R_1 = R'_1$ is at most $1/2^k$. Since there are $t^2$ pairs of inputs, the probability that *any two of them* will yield a collision in the right half is at most $t^2/2^k$, which is negligible (since $t$ is polynomial). To summarize, World 1 and World 2 are indeed indistinguishable.

We now go one more level down. World 3 is the same as level 2, except the value $R_2$, instead of being equal to $f_2(R_1) \oplus L_1$, is always chosen random. However, this *does not change the experiment at all*: namely, World 2 and World 3 are *the same*. Indeed, since World 2 only runs when no two values $R_1$ are the same, all $t$ inputs to $f_2$ are distinct. And since $f_2$ is a truly random function, all the values $f_2(R_1)$ are random and independent from each other. But, then irrespective of which values of $L_1$ are used, all $R_2 = L_1 \oplus f_2(R_1)$ are random and independent as well.

Next, we go to World 4. It is the same as world 3 up to the last round of the Feistel. There, it sets $L_3 = R_2$, as it should, but selects a truly random $R_3$ instead of expected $f_3(R_2) \oplus L_2$. First, since $R_2$ was completely random in World 3, $L_3 = R_2$ is completely

random as well. Next, by birthday paradox analysis, the probability that any two values $R_2$ (which are randomly selected) collide, is at most $t^2/2^k$, which is negligible. Thus, with overwhelming probability all the values $R_2$ are distinct. But then, since $f_3$ is a random function, by similar argument to the previous paragraph we get that all the values $f_3(R_2)$ are random and independent, and thus, so are $R_3 = f_2(R_2) \oplus L_2$. To summarize, with all but negligible probability World 3 and World 4 are the same as well.

Finally, notice that World 4 is a truly random function provided no right half collision happens on the first layer (in which case we stop the experiment). But the latter has negligible chance as we know, so World 4 is indeed indistinguishable from a random function. This completes the proof. □

We conclude our discussion with another useful theorem, which will not be proven here.

**Theorem 5 (PRF⇒ Strong PRP)** *Let $\mathcal{F}$ be a* PRF *family whose functions map $k$ bits to $k$ bits (i.e. $\ell(k) = L(k) = k$). Then $\mathcal{G} = \mathcal{H}^4_{\mathcal{F}}$ is an efficient* strong PRP *family. Namely,* 4 *rounds of the Feistel Network yield an efficient strong* PRP.

# 4 Using PRP's for Secret-Key Encryption

## 4.1 Several Schemes Using PRP's

In these section we propose several schemes (called ciphers) using PRP's. In all of them, let $\mathcal{G} = \{g_s\}$ be an efficient PRP family operating on the appropriate input length. The seed $s$ will always be part of the shared secret key.

**Electronic Code Book (ECB) Cipher.** This is the naive suggestion we started from:

$$c = E_s(m) = g_s(m); \qquad m = D_s(c) = g_s^{-1}(m)$$

We noted that ECB cipher is length-preserving and very efficient. However, it is obviously not CPA-secure, since the encryption of a message is completely deterministic (e.g. the adversary can tell if you send the same message twice). On the positive side, it is clearly one-message secure.

Next, we examine two obvious schemes resulting when we treat our PRP family as a PRF family (which is justified by Theorem 3).

**Counter (CNT) Cipher.** This is a stateful deterministic encryption. Players keep the counter value cnt which they increment after every encryption/decryption.

$$c = E_s(m) = g_s(\mathsf{cnt}) \oplus m; \qquad m = D_s(c) = g_s(\mathsf{cnt}) \oplus c$$

The CPA-security of this scheme was shown when we talked about PRF's.

**XOR (aka R-CNT) Cipher.** This is a stateless probabilistic encryption.

$$(r, c) \leftarrow E_s(m) = (r, g_s(r) \oplus m); \qquad m = D_s(r, c) = g_s(r) \oplus c$$

where $r$ is chosen at random per each encryption. The CPA-security of this scheme was shown when we talked about PRF's.

Next, we give several ciphers which really use the inversion power of PRP's. We start with what we call "nonce" ciphers, but first make a useful digression. The term *nonce* means something that should be unique (but possibly known to the adversary). For example, the counter and XOR schemes above effectively used a nonce to encrypt $m$ via $c = g_s(R) \oplus m$. Indeed, to analyze the security of the above schemes, we only cared that all the $R$'s are *distinct*. For the counter scheme we ensured it explicitly by using the state cnt, while for the XOR scheme we selected $R$ at random from a large enough domain, and argued the uniqueness with very probability (using the birthday paradox analysis). Similarly, the nonce ciphers below use nonce $R$ as follows ($\circ$ denotes a clearly marked concatenation):

$$c \leftarrow E_s(m) = g_s(m \circ R); \qquad D_s(c): \text{ get } m \circ R = g_s^{-1}(c), \text{ output } m$$

Assuming all the nonces $R$ are unique, the CPA-security of this "scheme" is easy. Depending whether we generate nonces using counters+state or at random, we get:

**C-Nonce Cipher.** This is a stateful deterministic encryption. Players keep the counter value cnt which they increment after every encryption/decryption.

$$c \leftarrow E_s(m) = g_s(m \circ \mathsf{cnt}); \qquad D_s(c): \text{ get } m \circ \mathsf{cnt} = g_s^{-1}(c), \text{ output } m$$

**R-Nonce Cipher.** This is a stateless probabilistic encryption.

$$c \leftarrow E_s(m) = g_s(m \circ r); \qquad D_s(c): \text{ get } m \circ r = g_s^{-1}(c), \text{ output } m$$

where $r$ is chosen at random per each encryption.

**Random IV (R-IV) Cipher.** It operates as follows:

$$(r, c) \leftarrow E_s(m) = (r, g_s(r \oplus m)); \qquad m = D_s(r, c) = g_s^{-1}(c) \oplus r$$

where $r$ is chosen at random per each encryption. $r$ is sometime s called the "Initialization Vector" (IV). Notice the similarity with the R-CNT cipher: R-CNT sets $c = g_s(r) \oplus m$, while R-IV sets $c = g_s(r \oplus m)$. Not surprisingly, the proof for CPA-security of R-IV is also very similar to R-CNT, with a slight extra trick.

**Proof Sketch:** Rather than arguing that all the $r$'s are distinct w.h.p., as we did for R-CNT, we argue that all $(r \oplus m)$'s are distinct w.h.p. Indeed, irrespective of how the adversary chooses the message $m$ to be encrypted, $r$ is chosen at random, so $(r \oplus m)$ is random and independent from everything else. Hence, by birthday paradox analysis w.h.p. all $(r \oplus m)$'s are distinct, so our PRP (modeled as a truly random *function*, not permutation, for the analysis; see Theorem 3 for justification) operates on different inputs, so all the values $g_s(r \oplus m)$ are random and independent from each other. Thus, in particular, w.h.p. the challenge ciphertext is a random string independent from the bit $b$ used to select $m_0$ or $m_1$ to be encrypted. □

**Remark 4** *Notice, R-IV scheme does* not *work with counters instead of random strings. The corresponding insecure scheme C-IV would be the following (where* $\mathsf{cnt}$ *is incremented after each encryption/decryption):*

$$c \leftarrow E_s(m) = g_s(\mathsf{cnt} \oplus m); \qquad m = D_s(c) = g_s^{-1}(c) \oplus \mathsf{cnt}$$

*Looking at the analysis above, try to see what goes wrong!*

## 4.2 Block Ciphers and Modes of Operation

In the above schemes we assumed that we can choose a $\mathsf{PRP}$ whose input length is roughly the same as the length of the message. Hence, the longer is the message, the longer should $\mathsf{PRP}$'s input be. In practice, this is quite inconvenient. Instead, people usually design *fixed length* $\ell$ $\mathsf{PRP}$'s (say, $\ell = 128$ bits), just long enough to avoid brute force attacks. Then to encrypt a long message $m$, $m$ is split in to *blocks* $m_1 \dots m_n$, each of *block length* $\ell$, and the $\mathsf{PRP}$ is somehow used to encrypt these $n$ blocks. For this reason, efficient $\mathsf{PRP}$'s with fixed (but large enough) block length are called *block ciphers*. The block cipher itself is assumed (or modeled) to be an efficient $\mathsf{PRP}$. However, the main question is how to really use this block cipher to "combine together" the $n$ message blocks $m_1 \dots m_n$. Each specific such method is called *mode of operation*. As we will see, there are many modes of operation for block cipher — some secure and some not. The main goal of a "good" mode, aside from being secure, is to minimize the number of extra blocks output. Usually, at most one extra block/value is considered "efficient", but there are many other important criteria as well.

We now go through the previous "single" block schemes, and see how they yield a corresponding mode of operation.

**Electronic Code Book (ECB) Mode.** Simply apply the ECB cipher block by block: $c_i = g_s(m_i)$, for all $i$. Decryption is obvious. This mode is extremely efficient, length-preserving, but totally insecure.

**Counter (C-CTR) Mode.** Simply apply the C-CTR cipher viewing $m_1 \dots m_n$ as "separate messages". Since C-CTR is $\mathsf{CPA}$-secure, we know that the resulting scheme is $\mathsf{CPA}$-secure as well: $c_i = g_s(\mathsf{cnt} + i - 1) \oplus m_i$. At the end, update $\mathsf{ctr} = \mathsf{ctr} + n$. Here and everywhere, the addition is modulo $2^\ell$. Notice, the encryption is length-preserving, but keeps state as the penalty. Decryption is obvious. Notice, this works (even better) with $\mathsf{PRF}$'s, and not only $\mathsf{PRP}$'s. Also, the scheme is parallelizable both for encryption and for decryption.

The next three modes are motivated by the XOR (or R-CTR) cipher. Recall, it returns $(r, g_s(r) \oplus m)$. A straightforward way to iterate this cipher is to choose a brand new $r_i$ for every block $m_i$. However, this requires to send all the $r_i$'s, which doubles the length of the ciphertext.

**Random Counter (R-CTR) Mode.** This modes "combines" the idea of C-CTR and R-CTR modes. As a result, it avoids state, but yields only one "extra block". Encryption picks a random $r$ for every message, and sets $c_i = g_s(r+i-1) \oplus m_i$. It outputs $(r, c_1, \dots, c_n)$,

Decryption is obvious: $m_i = g_s(r+i-1) \oplus c_i$. A similar analysis to R-CTR cipher shows that with high probability, all $n$-tuples $(r, r+1, \ldots, r+n-1)$ do not overlap, so no collision occurs. Notice, this works (even better) with PRF's, and not only PRP's. Also, the scheme is parallelizable both for encryption and for decryption.

**Cipher Feedback (CFB) Mode.** This mode uses a different idea to get "fresh" randomness. After selecting an initial IV $r$, it sets $c_1 = g_s(r) \oplus m_1$, and then uses $c_1$ itself as the next $r$! Namely, $c_2 = g_s(c_1) \oplus m_2$. Intuitively, since $g_s$ looks like a random function (by Theorem 3), $c_1$ indeed "looks random" and independent from the original $r$, so we can use it to encrypt $m_2$. And so on. To summarize, set $c_0 = r$ and $c_i = g_s(c_{i-1}) \oplus m_i$ and output $(c_0, c_1 \ldots c_n)$. To decrypt, recover $m_i = g_s(c_{i-1}) \oplus c_i$. Notice, this works (even better) with PRF's, and not only PRP's. Also, the scheme is parallelizable for decryption, but not for encryption.

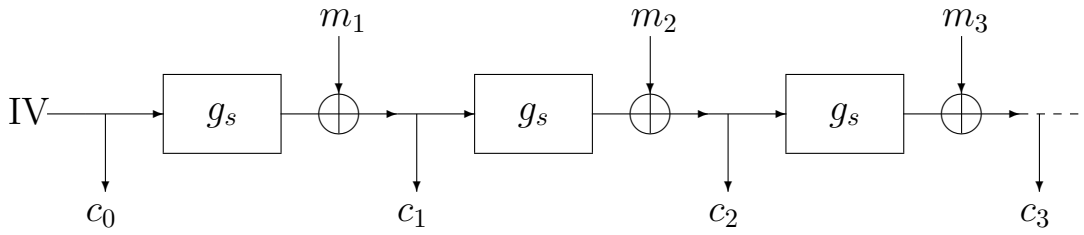



Figure 3: Cipher Feedback Mode

**Output-Feedback (OFB) Mode.** This mode also uses the block cipher itself to refresh randomness. Specifically, it picks a random IV $r$, and sets $r_0 = c_0 = r$. then, for each $i$, is sets "one-time pad" $r_i = g_s(r_{i-1})$ and computes $c_i = r_i \oplus m_i = g_s(r_{i-1}) \oplus m_i$ (the latter could also be viewed as the XOR cipher using $r_{i-1}$ as the IV). It outputs $(c_0, c_1 \ldots c_n)$. To decrypt, compute the $r_i$'s from $r_0 = c_0$ (via $r_i = g_s(r_{i-1})$) and output $m_i = c_i \oplus r_i = c_i \oplus g_s(r_{i-1})$. In other words, the initial IV $r$ defines a sequence of "independently looking" one-time pads: $r_1 = g_s(r)$, $r_2 = g_s(g_s(r))$, and so on: each $r_i$ is used to encrypt $m_i$. Notice, this works (even better) with PRF's, and not only PRP's. Also, the scheme is not parallelizable for either encryption, or decryption.

**Nonce Modes.** Those could be defined, both for the counter C-Nonce and the random R-Nonce versions, but are never used. The reason is that the overall size of encryption is by a constant fraction (rather than by one block length) larger than the length of the message, since a nonce should be used for each block. Notice, btw, *all* nonces have to be distinct.

**Cipher Block Chaining (CBC) Mode.** Finally, we define the last mode originating from the R-IV cipher. It is the only one so far that uses the inversion power of block ciphers. Recall, R-IV returns $(r, c)$, where $c = g_s(r \oplus m)$. A straightforward way to iterate this scheme is to use a different $r_i$ for each $m_i$ (check why one $r$ does not suffice!). Again, this would double the length of the encryption. Similarly to the CFB mode above, the

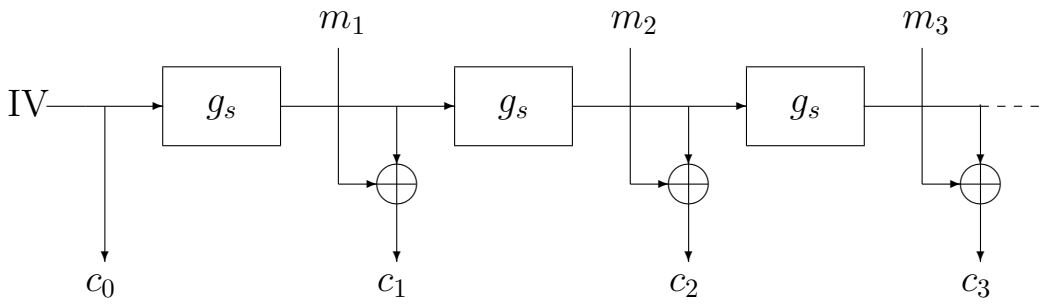



Figure 4: Output Feedback Mode

CBC mode uses the *previous cipher blocks* themselves as the new $r_i$'s, but uses those in R-IV rather than R-CTR! Specifically, as before it picks a random IV $c_0 = r$, and sets $c_1 = g_s(c_0 \oplus m_1)$. Then, it simply uses $c_1$ as the next IV: $c_2 = g_s(c_1 \oplus m_i)$. Then it uses $c_2$ as the next IV, and so on. To summarize, set $c_0 = r$, $c_i = g_s(c_{i-1} \oplus m_i)$ and output $(c_0, c_1 \dots c_n)$. To decrypt, recover $m_i = g_s^{-1}(c_i) \oplus c_{i-1}$. Notice, the scheme is parallelizable for decryption, but not for encryption.

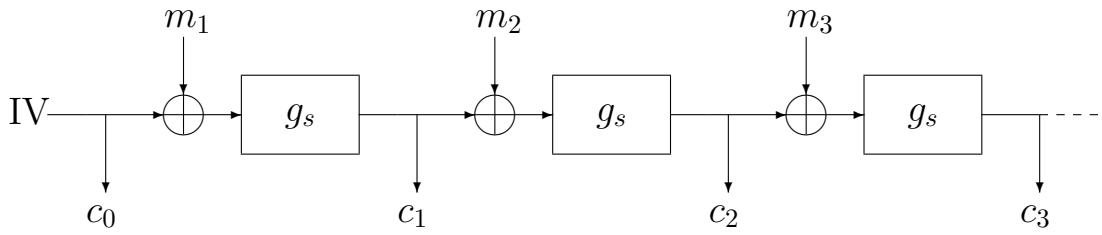



Figure 5: Cipher Block Chaining Mode

To summarize, C-CTR, R-CTR, CFB, OFB, CBC modes are all CPA-secure. For mysterious reason, CBC seems to be the most pervasively used mode in practice, despite being one of the lesser simple and secure of the above.

**Remark 5** *One can also define stateful variants of CFB, OFB and CBC modes, denoted C-CFB, C-OFB and C-CBC, where the initialization vector IV is set to be a counter instead of being chosen fresh for each encryption. It turns out that the modes C-CFB and C-OFB are still* CPA*-secure, which C-CBC is not! Think why this is the case (also read the next section).*

| CSCI-GA.3210-001<br>MATH-GA.2170-001 | **Introduction to Cryptography** | March28, 2012 |
|---|---|---|

Lecture 10

*Lecturer: Yevgeniy Dodis* *Spring 2012*

Last time we defined several modes of operation for encryption. Today we prove their security, and then spend the rest of the lecture studying the problem of message authentication. We will see that the problem is very different from encryption, although some tool, such as PRFs, come up pretty handy!

# 1 Security of CBC, CFB and OFB Modes

The CPA-security of the CFB, OFB and CBC modes is shown by very similar technique. In fact, one "universal proof" suffices. We give a proof sketch here.

First, as usual, the block cipher (i.e., our PRP) $g_s$ is replaced by a truly random function $F$ (this was justified last time) by first replacing it by a random permutation, and then replacing the latter by a random function). Now, take any adversary Adv. Rather than choosing the whole random function $F$ right away for the experiment with Adv, we select it "as needed" as we move along. Namely, we keep the table of $(input, output)$ pairs that were used so far. When a new input $x$ to $F$ arrives, we use the table if the input was used before, and otherwise add $(x, random)$ to the table, and use $random$ as the output of $F$. It is easy to see from the description of the CFB, OFB and CBC modes, that as long as we never perform the table look-up, i.e. no input is repeated twice during the run, the advantage of Adv is 0. Namely, all Adv sees is a bunch of random strings, independent from anything else.

Thus, it suffices to show that the probability an "input collision" happens is negligible. Let $B_j$ denote the ("bad") event that the input collision happened within first $j$ evaluations of $F$. We want to show that $\Pr(B_T) = \mathsf{negl}(k)$, where $T$ is the total number of calls to $F$ (roughly $T = nt$, where $t$ is the number of encryption queries made by Adv and $n$ is the number of blocks per message). We will show by induction on $j$ that

$$\Pr(B_j) \le \frac{(j-1)T}{2^\ell}$$

Indeed, $\Pr(B_1) = 0$. Now, in order for $B_j$ to happen, either there should be a collision among the first $(j-1)$ calls to $F$ (i.e., $B_{j-1}$ should happen), or the $j$-th input to $F$ collided with one of the previous $(j-1)$ inputs. Formally,

$$\Pr(B_j) \le \Pr(B_{j-1}) + \Pr(B_j \mid \overline{B_{j-1}})$$

Notice, by inductive assumption the first probability is at most $(j-2)T/2^\ell$. On the other hand, from the facts that (1) $\overline{B_{j-1}}$ means that the previous $(j-1)$-st input was distinct from all the earlier inputs; (2) $F$ is the random function, which combined with (1) gives that $(j-1)$-st output is truly random; and (3) the description of OFB/CFB/CBC modes

together with (1) and (2) implies that the $j$-th input is totally random, we conclude that the probability that *random* $j$-th input to $F$ collides with the previous $(j-1)$ inputs is at most $\frac{j-1}{2^\ell} \leq \frac{T}{2^\ell}$. Combining these,

$$\Pr(B_j) \leq \frac{(j-2)T}{2^\ell} + \frac{T}{2^\ell} = \frac{(j-1)T}{2^\ell}$$

But now we see that $\Pr(B_T) \leq T^2/2^\ell$, which is indeed negligible since $T$ is polynomial. Namely, w.h.p. no input collisions happen, and all the adversary simply sees is a bunch of truly random strings, giving him no way to predict the challenge $b$.

**Remark 1** *Notice, how more subtle the proof becomes for these modes. In some sense, it is surprising that much simpler C-CTR and R-CTR modes are not used more often: unlike OFB,CFB and CBC modes, they are easily parallelizable for both encryption and decryption; unlike the CBC mode they work with a* PRF *and not only a* PRP*; finally, they give comparable (or even better in case of C-CTR) security guarantees.*

## 2 Secret-Key Encryption in Practice

In practice, specific efficient block and stream ciphers are used. These practical schemes usually follow some high level guidelines we saw in our formal study. However, the design of block and stream ciphers is usually viewed as "black art". We name a few such ciphers used in practice, and refer you to outside sources to learn more about those (i.e., see "Applied Cryptography" by Bruce Schneier).

- Stream ciphers: RC4, SEAL, WAKE, A5, Hughes XPD/KPD, and many-many others.
- Block Ciphers: DES, AES (Rijndael), Lucifer, FEAL, IDEA, RC2, RC6, Blowfish, Serpent, and many-many-many-many others.

We notice that block ciphers are used much more commonly. Most of them (with a notable exception of AES) use Feistel Network as part of their design, so Feistel transform is very useful. The current standard is AES — advanced encryption standard, which recently replaced DES — old data encryption standard. Also, using any stream cipher, or a block cipher with a good mode of operation, the size of the ciphertext is (almost) equal to the size of the plaintext, which is very efficient.

## 3 Public-Key Encryption in Practice

In general, secret-key (or symmetric) encryption is very efficient (even the theoretical ones we constructed using PRGs). On the contrary, public-key (or asymmetric) encryption is much less efficient, especially for large messages (aside from efficiency issues, the size of the ciphertext is usually much larger than the plaintext as well). On the other hand, a major drawback to symmetric key schemes is that they require that you share a secret with the intended recipient of your communication. On the contrary, public key encryption does *not* require you to share any secret knowledge with the recipient.

So can we have a relatively efficient public-key encryption, especially for long messages? The answer is "yes". And the technique is to combine the "slow" ordinary public-key encryption $E'$ with the fast secret-key encryption $\mathsf{Enc}$. This folklore technique is known as integration of public- and secret-key schemes. If auxiliary $E'$ has public key $PK$ and secret key $SK$, we define new public-key encryption $E$, which has the same public/secret keys, as follows:

$$E_{PK}(m) = (E'_{PK}(s), \mathsf{Enc}_s(m)), \qquad \text{where } s \leftarrow_R \{0,1\}^k$$
$$D_{SK}(c_1, c_2): \quad \text{get } s = D'_{SK}(c_1); \quad \text{output } m = \mathsf{Dec}_s(c_2)$$

That is, we encrypt a short, randomly chosen secret using $s$ a public key scheme, and then transmit a stream or block cipher encryption of the message using the random secret. The recipient first decrypts the random secret, then uses that to decrypt the message. Provided that $E'_{PK}$ and $\mathsf{Enc}$ are $\mathsf{CPA}$-secure, the new scheme is also $\mathsf{CPA}$-secure (in fact, one-message security suffices for the symmetric scheme $\mathsf{Enc}$). The proof is left as an exercise.

In fact, this method can be generalized to any key encapsulation scheme capable of encapsulating a key $s$ long enough to perform symmetric encryption of long messages. If $\langle \psi, s \rangle$ is the output of the key encapsulation algortihm, simply set the ciphertext to $c = \langle \psi, \mathsf{Enc}_s(m) \rangle$. When decrypting $c$, use key decapsulation to recover $s$ from $\psi$, and then use $s$ to recover $m$.

# 4 Introduction to Authentication

The rest of the lecture we will be dedicated to the problem of *message authentication*. First, we define the problem of message authentication and show how it is different from the problem of encryption. Then we define the notion of message authentication schemes, and the sub-case of those — message authentication codes ($\mathsf{MAC}$). We will examine goals and capabilities of the intruder and will define the strongest security notion for $\mathsf{MAC}$s. Then we will see how $\mathsf{MAC}$ can be implemented using $\mathsf{PRF}$ (and the other way around). Then we will examine the problem with long messages and two approaches solving it. The second approach will lead us to the definition of a special family of *universal* hash functions, and the usage of such family to enhance the efficiency of $\mathsf{PRF}$ and $\mathsf{MAC}$ for long inputs.

## 4.1 Motivation

While Encryption is a topic of cryptography that deals with privacy, Authentication is a topic of cryptography that deals with trust. When a sender sends a message to a receiver, how does a receiver know if it comes from appropriate sender? What if the intruder tries to imitate a sender, or tampers with the real messages being sent, etc.? Is there a way for the recipient to be "sure" that the message indeed came from the supposed sender, and was not modified in the transit?

Similar to the encryption scenario, there are two approaches to solving this problem: the *secret-key* and the *public-key*. In this lecture we start with the secret-key setting. In this scenario, the sender and the recipient share a secret key $s$ (not known to the attacker). This key helps the sender to "tag" the message, so that the recipient (only) can verify the

validity of the "tag", but nobody can "forge" a valid tag. Thus, the goal is to establish authenticated communication between a dedicated sender/receiver pair.

To summarize, we are trying to design the following "$s$-functionality". Let's say sender is to send message $m$ (in the clear, we are not solving encryption problem at the moment). So, it calls up his $s$-functionality that put the message $m$ into the "envelope" $T$. Then it goes through open communication, accessible to intruders, so by the time it gets to the receiver it could change from $T$ to some $T'$. Receiver receives this envelope $T'$. He calls up his $s$-functionality that validates whether envelope $T'$ was stuffed by $s$-functionality on the sender's side. If it does, we can be "sure" that $T' = T$, receiver can extract $m'$ from $T'$ as a valid authenticated message, and in fact $m' = m$. Otherwise receiver rejects $T'$ as invalid. Thus, the security of Authentication would imply inability by the intruder to produce a valid $T'$ (in other words to send to the receiver something it would accept), not produced by the sender. The above is very close to a formal definition of a *message authentication scheme*.

## 4.2 Message Authentication Schemes

The above discussion leads to the following general definition (below we only define the syntax, and not the *security*). Below $\mathcal{M}$ is the corresponding message space, e.g. $\mathcal{M} = \{0,1\}^\ell$.

Definition 1 [Message Authentication Scheme] Message Authentication Scheme is a triple ($\mathsf{Gen}$,$\mathsf{Auth}$,$\mathsf{Rec}$) of $\mathsf{PPT}$ algorithms:

a) The key generating algorithm $\mathsf{Gen}$ outputs the shared secret key: $s \leftarrow \mathsf{Gen}(1^k)$.

b) The message authentication algorithm $\mathsf{Auth}$ is used to produce a value ("envelop") $T \leftarrow \mathsf{Auth}_s(m)$, for any $m \in \mathcal{M}$. It could be deterministic, as we will see.

c) The deterministic message recovery algorithm $\mathsf{Rec}$ recovers the message from the envelope $T$: $\mathsf{Rec}_s(T) = \tilde{m} \in \mathcal{M} \cup \{\perp\}$, where $\perp$ means that the message was improperly authenticated.

The correctness property states that $\tilde{m} = m$, i.e. $\forall s, m, \quad \mathsf{Rec}_s(\mathsf{Auth}_s(m)) = m$. $\diamondsuit$

Let us for now assume our message authentication schemes are stateless. Later we can examine stateful message authentication schemes as well.

Before moving any further (in particular, define the security of message authentication schemes), let us compare (secret-key) encryption and authentication.

## 4.3 Authentication vs. Encryption

Just to see how different the problem of Authentication is from the problem of Encryption, let's see that Encryption is not solving (and is *not intended to solve*) authenticity at all. Even the best encryption schema, one time pad for one message, was not addressing the issue. Lets say intruder $E$ knows format of the message where the sender says "please credit 100 dollars to $E$", and say the one-time pad is used over ASCII alphabet. He can easily "flip" the 1 for a 9 for example, by simply flipping several bits of the encrypted message (in fact, the encryption will reveal the one-time pad, and $E$ can encrypt anything he wants

with it now). Similar attacks will apply to other encryption schemes we studied so far (e.g., counter scheme will have the same attack).

To summarize, encryption schemes are designed so that $E$ cannot *understand* the contents of the encryption, but might allow $E$ to tamper with it, or insert bogus messages. For example, in many encryption schemes *every* string is a legal encryption is some valid message. In this case, $E$ can send any "garbage" string, and the recipient will decrypt it into a valid (albeit probably "useless") message. This should not be possible in a "secure" message authentication scheme.

Conversely, authentication be itself does not solve privacy. In fact, nothing prevents the envelop to have the message $m$ *in the clear*.[1] To emphasize this point even further, for the rest of this lecture we will talk about a special class of message authentication schemes, called *message authentication codes* (MACs), which *always* contain the message in the clear. As we will see, dealing with this special case is more intuitive, since it clearly illustrates our goal of message authentication.

# 5 Message Authentication Codes (MAC)

## 5.1 Definition

As we said, MAC is a special case of message authentication schemes. It implies sending message $m$ in the clear along with a *tag* $t$. Namely, envelope $T = (m, t)$, and the message recovery only checks if the tag $t$ is "valid". Thus, the receiver's goal is to validate message $m$ by verifying whether she/he can trust that tag $t$ is a valid tag for $m$. In case of successful validation receiver outputs "accept", otherwise "reject".

Definition 2 [Message Authentication Code] Message Authentication Code, MAC, is a triple (Gen,Tag,Ver) of PPT algorithms:

a) The key generating algorithm Gen outputs the shared secret key: $s \leftarrow \mathsf{Gen}(1^k)$.

b) The tagging algorithm Tag is produces a tag $t \leftarrow \mathsf{Tag}_s(m)$, for any $m \in \mathcal{M}$. It could be deterministic, as we will see.

c) The deterministic verification algorithm Ver produces a value $\mathsf{Ver}_s(m, t) \in \{accept, reject\}$ indicating if $t$ is a valid tag for $m$.

The correctness property states that $\tilde{m} = m$, i.e. $\forall s, m, \quad \mathsf{Ver}_s(m, \mathsf{Tag}_s(m)) = accept$. $\diamondsuit$

## 5.2 Security

The security as usual is measured by the probability of unauthorized PPT adversary $\mathcal{A}$ to succeed. But what is the goal of $\mathcal{A}$? The most ambitious goal is to recover secret key $s$. Similar to the encryption scenario, this is too ambitious (e.g., part of $s$ can be never used, so it is not a big deal to prevent $\mathcal{A}$ from recovering $s$). A more reasonable goal is to come up with the message-tag pair $(m, t)$ such that $\mathsf{Ver}_s(m, t) =$ "*accept*". Having that pair, $\mathcal{A}$

[1] Later we will study the problem of *authenticated encryption*, which simultaneously provides privacy and authenticity.

can masquerade as a valid sender of $m$ by simply sending $(m, t)$ to the receiver. This attack is called a *forgery*. There are two kinds of forgery: universal and existential. Universal forgery implies that $\mathcal{A}$ can come up with trusted tag $t$ for any message $m$ that $\mathcal{A}$ wants. Existential forgery implies that $\mathcal{A}$ can produce at least a single pair $(m, t)$ that triggers $\mathsf{Ver}$ to "accept", even if the message $m$ is "useless". Existential forgery is the least ambitious of these goals, so the strongest security definition would imply that no $\mathcal{A}$ can achieve even this, easiest of the goals. Moreover, in many applications such (strong) security is indeed needed.

Next, what can $\mathcal{A}$ do to try to achieve its goal? There are several options.

- The weakest attack mode is to give $\mathcal{A}$ no special capabilities. I.e., $\mathcal{A}$ cannot intrude into communication between sender and receiver and has no oracle access to any of $\mathsf{MAC}$ algorithms.

- A more reasonable assumption is that $\mathcal{A}$ has some access to communication between sender and receiver, and can try to analyze valid message-tag pairs $(m, t)$ that it observes. Of course, once we allow this, we have to restrict $\mathcal{A}$ from outputting a "forgery" $(m, t)$ for the message $m$ that it already observed.[2] Taken to the extreme, we can allow $\mathcal{A}$ observe valid tags for *any* message $m$ that $\mathcal{A}$ wants. Namely, we can give $\mathcal{A}$ *oracle access* to the tagging function $\mathsf{Tag}_s(\cdot)$. This is called (adaptive) *chosen message attack* ($\mathsf{CMA}$).

- Finally, we can assume $\mathcal{A}$ has oracle access to the receiver, so that it can learn whether any message-tag pair $(m, t)$ is valid. Alternatively, we may think that $\mathcal{A}$ keeps sending "fake" message/tag pairs, and wins the moment the recipient accepts such a pair.

Looking at the above, we now make the strongest definition of security, such that prevent $\mathcal{A}$ from its easiest goal, existential forgery, but lets $\mathcal{A}$ use both the tagging and the verification oracle.

DEFINITION 3 [$\mathsf{MAC}$'s Security] A $\mathsf{MAC} = (\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$ is "secure", i.e. existentially unforgeable against $\mathsf{CMA}$, if $\forall$ $\mathsf{PPT}$ $\mathcal{A}$, who outputs a "forgery" $(m, t)$ such that $m$ has never been queried to oracle $\mathsf{Tag}_s(\cdot)$,

$$\Pr(\mathsf{Ver}_s(m, t) = \mathit{accept} \mid s \leftarrow \mathsf{Gen}(1^k),\ (m, t) \leftarrow \mathcal{A}^{\mathsf{Tag}_s(\cdot), \mathsf{Ver}_s(\cdot)}(1^k)) \leq \mathsf{negl}(k)$$

$\diamondsuit$

For completeness, we also give the corresponding (more general definition) for any message authentication scheme (even though we will mainly work with $\mathsf{MAC}$s).

[2] A slightly stronger definition forbids $\mathcal{A}$ to output a *pair* $(m, t)$ that it already observed. In other words, $\mathcal{A}$ is allowed to forge a "new" tag $t'$ for the message $m$ whose tag $t \neq t'$ it already observed. This notion is called *strong unforgeability*. Luckily, most of our constructions will be strongly unforgeable. Moreover, in most (but not all; see below) settings this strengthening is not that crucial. First, most $\mathsf{MAC}$'s are deterministic and have "canonical" verification $t \stackrel{?}{=} \mathsf{Tag}_s(m)$, in which case there is no difference between the standard and strong unforgeability. Second, if $m$ was legally sent and the receiver is "allowed" to accept $m$ again, it will do it with the "old" tag $t$ as well, so there is no value to even change $t$ to $t' \neq t$. Finally, in many applications such "duplication" is anyway forbidden and is enforced by other means like time-stamps, etc.

DEFINITION 4 [Security of Message Authentication Scheme] A message authentication scheme $(\mathsf{Gen}, \mathsf{Auth}, \mathsf{Rec})$ is "secure", i.e. existentially unforgeable against CMA, if $\forall$ PPT $\mathcal{A}$, who outputs a "forgery" $T$ such that $\mathsf{Rec}_s(T)$ has never been queried to oracle $\mathsf{Auth}_s(\cdot)$,

$$\Pr(\mathsf{Rec}_s(T) \neq \perp \mid s \leftarrow \mathsf{Gen}(1^k),\ T \leftarrow \mathcal{A}^{\mathsf{Auth}_s(\cdot),\mathsf{Rec}_s(\cdot)}(1^k)) \leq \mathsf{negl}(k)$$

$\diamondsuit$

**Remark 2**

*a) Usually,* Gen *just outputs a random string as a key. We will see examples of this later.*

*b) When* Tag *is deterministic and verification* Ver *is "canonical" (i.e. it simply checks* $\mathsf{Tag}_s(m) \stackrel{?}{=} t$*), — which include the deterministic PRF-based MACs studied in the next section, — oracle access to* $\mathsf{Ver}_s(\cdot)$ *is not needed.*[3] *Instead of calling* $\mathsf{Ver}_s(m', t')$*, $\mathcal{A}$ can call* $\mathsf{Tag}_s(m')$ *and compare its output with* $t'$*. And if $\mathcal{A}$ would like to check his candidate forgery* $(m', t')$ *before outputting it, and at most $q$ such checks are made by $\mathcal{A}$, we can simply let $\mathcal{A}$ "pretend" that all the checks are false, and output at random one of these $q$ questions to* Tag *as its forgery. True, the success probability goes down by a factor of $q$, but since $q$ is polynomial in $k$, it still remains non-negligible if it was non-negligible. More generally, that last argument extends to general message authentication scheme: if* Auth *is deterministic and* Rec *is "canonical" (i.e., it also checks* $T \stackrel{?}{=} \mathsf{Auth}_s(\mathsf{Rec}_s(T))$*), oracle access to* $\mathsf{Rec}(\cdot)$ *is not needed (why?), even though again we loose a large polynomial factor in the security.*

## 5.3 MACs, Unpredictable Functions and PRFs

In this section we will build a secure MAC. In fact, we restrict our attention to *deterministic* (stateless) MAC's, whose keys ("seeds") are random strings of some length. Hence and without loss of generality, the verification algorithm $\mathsf{Ver}_s(m, t)$ simply checks if $t = \mathsf{Tag}_s(m)$, so we do not need to explicitly specify it. Moreover, $\mathsf{Tag}_s(m)$ is now a deterministic function indexed by the seed $s$. Thus, we can view such a MAC as a *family of functions* $\mathcal{F} = \{f_s\{0,1\}^\ell \rightarrow \{0,1\}^{|tag|}, s \leftarrow \mathsf{Gen}(1^{|s|})\}$, where $f_s = \mathsf{Tag}_s$. For simplicity, let us assume $|s| = k$. The chosen message attack (with no unnecessary oracle to $\mathsf{Ver}_s$) corresponds to oracle access to $f_s(\cdot)$, for a random (unknown) $s$. The success corresponds to outputting a correct pair $(x, f_s(x))$, where $x$ was not submitted to the $f_s(\cdot)$ oracle, i.e. *predicting* the value $f_s(x)$ for a "new" $x$. Not surprisingly, such, in some sense "canonical", MAC is called an *unpredictable function*. More formally, the family $\mathcal{F}$ is called an *unpredictable family* of for any PPT $\mathcal{A}$,

$$\Pr(f_s(x) = y \mid s \leftarrow \{0,1\}^k,\ (x, y) \leftarrow A^{f_s(\cdot)}(1^k)) \leq \mathsf{negl}(k)$$

where $x$ is not allowed to be queried to the oracle $f_s(\cdot)$. Hence, to construct our first secure MAC it suffices to construct an unpredictable function family. Not suprisingly, a PRF family is an unpredictable family, provided its output length $b$ is "non-trivial" (i.e., $\omega(\log k)$, so that $2^{-b} = \mathsf{negl}(k)$).

[3]The canonical verification is important. Otherwise, one can have artificial examples where carefully crafted verification queries on a message whose "canonical" tag is known to the attacker could reveal the entire secret key $s$.

**Theorem 1** (PRF $\Rightarrow$ MAC) *If $\mathcal{F}$ is a* PRF *family with non-trivial output length, then $\mathcal{F}$ is unpredictable, and thus defines a secure deterministic* MAC.

**Proof:** By the random function model, we can replace $f_s$ with a truly random function $f$ with output length $b$. Since $b$ is "non-trivial" and $x$ has to be "new", any adversary has a negligible probability (exactly $2^{-b}$ in the random function model) to predict $f(x)$, completing the proof. □

Hence, "pseudorandomness implies unpredictability". But does the converse hold? It turns out *the answer is "yes"*, but the construction is quite tricky. Given unpredictable $\mathcal{F}$, we construct the following family $\mathcal{G}$ with output length 1 bit (from the properties of PRF's, such "short" output is not a problem and can be easily stretched), very similar to the Goldrecih-Levin construction.

**Theorem 2 (Naor-Reingold, Goldreich-Levin)** *Let $\mathcal{F} = \{f_s : \{0,1\}^\ell \to \{0,1\}^b \mid s \in \{0,1\}^k\}$ be an unpredictable family. Define $\mathcal{G} = \{g_{s,r} : \{0,1\}^\ell \to \{0,1\} \mid s \in \{0,1\}^k, r \in \{0,1\}^b\}$ as follows: $g_{s,r}(x) = f_s(x) \cdot r \bmod 2$, where $\alpha \cdot \beta$ denotes the inner product modulo 2, for $\alpha, \beta \in \{0,1\}^b$. Then $\mathcal{G}$ is a* PRF *family.*

We remark that it is very important that the value $r$ be kept secret (as implied by the notation above). Notice, this is different from the usual Goldreich-Levin setting, where the anverter for the one-way function learns $r$. Also notice that we now know that OWF $\Rightarrow$ PRF $\Rightarrow$ MAC $\Rightarrow$ OWF, i.e. all these primitives are equivalent!

### 5.4 Dealing with Long Messages

In practice, concrete PRF have short fixed input length $\ell$ (for example, when implemented using a block cipher). One the other hand, in practice we want to be able to sign messages of length $L \gg \ell$ (e.g., one wants to sign a book using 128-bit block cipher modelled as a PRF). There are two approaches to deal with this problem.

1. Split $m$ into $n$ blocks $m_1 \dots m_n$ of length $\ell$ each (let's not worry about messages whose lenth is not a multiple of $\ell$). Somehow separately tag each block using $f_s$. The simplest approach would be to just output $f_s(m_1) \circ \dots \circ f_s(m_n)$ as a tag. However, one can easily see that this is insecure (why?). Using more advanced suggestion, one can try $f_s(1 \circ m_1) \circ \dots \circ f_s(n \circ m_n)$. Again, there is a problem (which?). Turns out that a few more "fixes" can make this approach work. Unfortunately, the length of the tag now is quite large (proportional to $nb = Lb/\ell$, which could be large). Can we make it smaller? This is what the second approach below does.

2. The second approach involves a general efficient construction of a PRF family on $L$ bit inputs from the one on $\ell \ll L$ bit inputs. Moreover, the output of the resulting PRF is as short as that of the original PRF (implying that tags as still very short!). Notice, this is more general than building a "long-input" MAC out of a "short-input" PRF: we actually build a "long-input" PRF!

   The idea is to design a special shrinking (hash) function $h : \{0,1\}^L \to \{0,1\}^\ell$ and use $f_s(h(m))$ as the tag. Notice, however, since there are $2^{L-\ell}$ times more possible

messages than there are possible hash value, there are many pairs of messages $(m_1, m_2)$ which "collide" under $h$: $h(m_1) = h(m_2)$. Unfortunately, the knowledge of any two such messages $m_1$ and $m_2$ allow the adversary to produce break the resulting "pseudo-PRF"; in fact, to produce a forgery of the resulting "pseudo-MAC". Indeed, since $f_s(h(m_1)) = f_s(h(m_2))$, $\mathcal{A}$ can ask for the tag $\alpha = f_s(h(m_1))$ of $m_1$ and output $(m_2, \alpha)$ as the forgery of a "new" message $m_2$. There are two ways out of this problem:

- First way is to notice that it suffices to ensure that *it is computationally hard* to find such a collision $(m_1, m_2)$ for $h$, even if the attacker knows the description of $h$. Such functions are called *collision-resistant* hash functions (CRHFs). It is easy to see that using CRHFs is easily enough to solve our problem. However, we will see that it is quite non-trivial to construct them.[4] We will come back to this approach soon.
- A simpler and more effective way to solve the problem is to use a *secret function* $h$ not known to the attacker. Formally, we need a *family of hash functions* $h$:

$$\mathcal{H} = \{h_t : \{0,1\}^L \to \{0,1\}^\ell, t \in \mathcal{K}\}$$

where $\mathcal{K}$ is the correspinding key space for $t$. Now, we pick two independent keys for our resulting family $\mathcal{F}(\mathcal{H})$: the key $s$ for $f_s$, and the key $t$ for $h_t$, i.e. $\mathcal{F}(\mathcal{H}) = \{f_s(h_t(\cdot))\}$. The main question is:

*What properties of $\mathcal{H}$ make $\mathcal{F}(\mathcal{H})$ is a* PRF *family when $\mathcal{F}$ is such?*

We answer this last question in the next section.

## 6 PRF and Universal Hashing

As the first observation, notice that for no two messages $m_1 \neq m_2$ can we have $\Pr_t(h_t(m_1) = h_t(m_2)) \geq \varepsilon$, where $\varepsilon$ is non-negligible. Namely, no two elements are likely to collide. Indeed, otherwise the adversary who learns $\alpha = f_s(h_t(m_1))$ has probability $\varepsilon$ that $f_s(h_t(m_2)) = \alpha$, which breaks the security of the PRF. It turns out that this necessary condition is also *sufficient*!

Definition 5 [$\varepsilon$-universal family of hash functions] $\mathcal{H}$ is called *$\varepsilon$-universal* if

$$\forall\ x, x' \in \{0,1\}^L,\ \text{s.t. } x \neq x' \implies \Pr_t(h_t(x) = h_t(x')) \leq \varepsilon$$

If $\varepsilon = 2^{-\ell}$ (which turns out the smallest it can get when $\ell < L$), then $\mathcal{H}$ is simply called (perfectly) *universal*. In assymptotic terms, $\mathcal{H}$ is called *almost universal (AU)* if $\varepsilon = \mathsf{negl}(|t|)$. ♢

**Theorem 3** *$\mathcal{F}(\mathcal{H}) = \{f_s(h_t(\cdot))\}$ is a* PRF *(and thus defines a* MAC*) if $\mathcal{F}$ is a* PRF *family and $\mathcal{H}$ is almost universal.*

[4] In particular, it turns out one is unlikely to construct them even from the strongest general assumption that we studied so far — existence of TDPs. However, we will later construct them from specific number-theoretic assumptions, such as discrete log.

**Proof:** We have to show that $\forall$ PPT $\mathcal{A}$, $\mathcal{A}^{f_s(h_t(\cdot))} \approx \mathcal{A}^{Z(\cdot)}$, where $Z$ is random function from $\{0,1\}^L$ to $\{0,1\}^b$. We will use two-step hybrid argument:

$$\mathcal{A}^{f_s(h_t(\cdot))} \approx \mathcal{A}^{R(h_t(\cdot))} \approx \mathcal{A}^{Z(\cdot)}$$

where $R$ is random function from $\{0,1\}^\ell$ to $\{0,1\}^b$. The first step immediately follows from the definition of PRF. Hence, it suffices to show that $\mathcal{A}^{R(h_t(\cdot))} \approx \mathcal{A}^{Z(\cdot)}$.

Let $m_1 \dots m_q$ be (wlog, distinct) queries $\mathcal{A}$ makes to its oracle (whatever it is). When given oracle access to $Z(\cdot)$, $\mathcal{A}$ gets $q$ totally random and independent values. Intuitively, since $R$ is a radnom function, as long as no two values $h_t(m_i)$ collide, $\mathcal{A}$ also gets $q$ totally random and independent values. Thus, it suffices to show that the probability $h_t(m_) = h_t(m_j)$ for some $i$ and $j$ is $\mathsf{negl}(k)$. Since $\mathcal{H}$ is $\varepsilon$-unversal, where $\varepsilon = \mathsf{negl}(k)$, and there are at most $q^2 \leq \mathsf{poly}(k)$ pairs of $i$ and $j$, the intuitive claim above follows, since $q^2\varepsilon = \mathsf{negl}(k)$.

The above intuition is almost formal, even though making it formal is slightly tricky. Let $X$ be the event that during $\mathcal{A}$'s run with $R(h_t)$-oracle, a collision happened among $h_t(m_1) \dots h_t(m_q)$. First, if $X$ does not happen, the values $R(h_t(m_1)) \dots R(h_t(m_q))$ are all random and independent from each other, exactly as the $Z$-oracle would return. Hence, the probability that $\mathcal{A}$ can tell apart the "$Z$-world" and the "$R(h_t)$-world" is at most the probability of $X$ (defined in the "$R(h_t)$-world"). However, and this is the tricky point, once a collision happens in "$R(h_t)$-world", it does not matter how we answer the oracle queries of $\mathcal{A}$, since $X$ has already happened!

Specifically, we can imagine the modified "$R(h_t)$-world", where *all* the queries of $\mathcal{A}$ are answered completely at random and independently from each other, irrespective of whether of not $X$ happened. We claim that this does not alter the probability of $X$. Indeed, up to the point a collision happened (if it ever happens), all the queries are supposed to be answered at random, since $R$ is a random function. What happens after $X$ happens is irrelevant. But now we run $\mathcal{A}$ in a manner *independent from* $t$. Indeed, all the queries are simply answered at random. Thus, we can imagine that we *first* ran $\mathcal{A}$ to completion, and *only then* selected $t$ and checked if $X$ (i.e., a collision among $h_t(m_1) \dots h_t(m_q)$) happened! But this means that $m_1 \dots m_q$ are defined *before* (and independently from) $t$. By the $\varepsilon$-universality of $\mathcal{H}$, and since there are at most $q^2$ pairs of indices $i < j$, we get that $\Pr(X) \leq q^2\varepsilon = \mathsf{negl}(k)$, since $q$ is polynomial and $\varepsilon$ is negligible. This argument completes the proof. $\square$

| CSCI-GA.3210-001<br>MATH-GA.2170-001 | **Introduction to Cryptography** | April 4, 2012 |
|---|---|---|

# Lecture 11

*Lecturer: Yevgeniy Dodis* *Spring 2012*

Last time we defined almost universal hash functions, and showed how they are useful for message authentication. Recall, such family $\mathcal{H} = \{h_t : \{0,1\}^L \to \{0,1\}^\ell\}$ has the property that for all $m \neq m'$, $\Pr_t(h_t(m) = h_t(m')) \leq \varepsilon$, where $\varepsilon$ is negligible in the secuirty parameters. We now give a variety of almost universal families $\mathcal{H}$. As will will see, this primitive is quite easy to construct, both information-theoretically and computationally. Then we proceed to study the resulting MACs, as well as several other ways to design MACs. Then we switch our attention to collision-resistant hash funstions.

## 1 Information-Theoretic Examples

**Inner Product Construction.** Let $F$ be a finite field of size roughly $2^\ell$. In particular, $F = GF[2^\ell]$ is most convenint, but $F = \mathbb{Z}_p$ is also OK for $p \approx 2^\ell$. View the message $m \in \{0,1\}^L$ as $n$ elements $m_1 \dots m_n$ of $F$, where $n \approx L/\ell$. For example, if $F = GF[2^\ell]$, we simply split the message into $\ell$-bit chucks $m_1 \dots m_n$ and view each block $m_i$ as an element of $GF[2^\ell]$.

The secret key $t$ of $h$ consists of $n$ elements $a_1 \dots a_n$ of $F$. Thus, the length of $t$ is (roughly) $L$, equal to the length of the message $m$. Now, define

$$h_{a_1 \dots a_n}(m_1, \dots, m_n) = \sum_{i=1}^{n} a_i m_i \qquad \text{(the operations are in } F\text{)}$$

Let us now examine the probability of a collision for any $m \neq u$. Let $z_i = m_i - u_i$. As $x \neq y$, at least one of the $z_i$ is a non-zero element of $F$. By symmetry and for the ease of notation, let us assume that this is $z_1 \neq 0$. Now, in order for $h_{a_1 \dots a_n}(m_1, \dots, m_n) = h_{a_1 \dots a_n}(u_1, \dots, u_n)$, we must have

$$\sum_{i=1}^{n} a_i m_i = \sum_{i=1}^{n} a_i u_i \quad \Leftrightarrow \quad a_1 z_1 = -\sum_{i=2}^{n} a_i z_i \quad \Leftrightarrow \quad a_1 = -\left(\sum_{i=2}^{n} a_i z_i\right)/z_1$$

Now, what is the probability that a random field element $a_1$ is equal to the last expression (whatever that experssion is, notice that the choice of $a_1$ is *independent* of it)? Clearly, it is $1/|F| \approx 2^{-\ell}$. In particular, it is the optimal value $2^{-\ell}$ when $F = GF[2^\ell]$. Thus, this construction achieves optimal $\varepsilon$, but the key length of $t$ is equal to $L$, which is too large. Instead, we would like the key to be $O(\ell)$, independent of the size $L$ of the message!

**Polynomial Construction.** As before, let $F$ be a finite field of size roughly $2^\ell$ (either $\mathbb{Z}_p$, or, more conveniently, $GF[2^\ell]$ since it takes exactly $\ell$ bits to represent an element in this field). As before, view $m = m_1, \dots, m_n$ (i.e., $|m| = L \approx n\ell$), where each $m_i \in F$. Now,

however, we view $m_1 \dots m_n$ as $n$ coefficients of a degree $(n-1)$ polynomial over $F$ (see below). We will also select a random point $x \in F$ as the key to a function $h_x$ in the hash family, defined as

$$h_x(m_1, \dots, m_n) = q_m(x) = \sum_{i=1}^{n} m_i \cdot x^{i-1}$$

where all the operations are done in $F$. Let's examine the probability of a collision between two distinct "polynomials" $m$ and $u$. A collision here means

$$h_x(m) = h_x(u) \Longleftrightarrow q_m(x) = q_u(x) \Longleftrightarrow q_{m-u}(x) = 0 \Longleftrightarrow \sum_{i=1}^{n} (m_i - u_i) \cdot x^{i-1} = 0$$

where at least one $m_i - u_i \neq 0$, i.e. $q_{m-u}(\cdot)$ is a *non-zero* polynomial of degree at most $(n-1)$. It is a well known fact that any *non-zero* polynomial of degree $d$ can have at most $d$ roots in $F$. Since the point (our key) $x \in F$ was chosen at random, the probability that $x$ is one of these at most $(n-1)$ roots of $q_{m-u}(\cdot)$ is at most $\frac{n-1}{|F|} \approx \frac{L}{2^\ell \ell}$, which is negligible.

Also, the key size is only $\ell$ bits, independent of the message length $L = n\ell$ (instead, the error depends on $L$). It turns out that one can achieve the best of both world — small key length and error probability close to $2^{-\ell}$. Concretely, one can acheive $|t| = O(\ell + \log N)$ and $\varepsilon = 2^{1-\ell}$. But we will not give this construction here.

## 2 Computational Examples (XOR-MAC, CBC-MAC, HMAC)

The next several examples use a PRF family $\mathcal{F} = \{f_t : \{0,1\}^\ell \to \{0,1\}^\ell\}$. Notice, we are slightly cheating here for 2 reasons. First, we are using "short-input" PRF $f_t$ to build "long-input" *computationally* almost universal $\mathcal{H} = \{h_t\}$. This means that for any PPT attacker who outputs two messages $x' \neq x'$, the probability that $h_t(x) = h_t(x')$ is negligible:

$$\Pr(h_t(x) = h_t(x') \mid t \leftarrow \$, (x, x') \leftarrow A(1^k)) = \mathsf{negl}(k)$$

This is luckily enough for our purposes (i.e., the composition of "short" PRF with computationally almost universal $\mathcal{H}$ still yields "long" PRF). But the reason this comes up is that in the analysis of $\varepsilon$-universaility we will immediately replace $f_t$ by a truly random function $R$. But this change means that the actual family we construct using $\mathcal{F}$ is only computationally almost universal.

Second, to build our "long-input" PRF, we will have to combine our $h_t$ constructed using $f_t$ with another *independently selected* PRF $f_s$, via $f_s(h_t(\cdot))$. As we will see, however, a simple general trick allows us to avoid making $s$ and $t$ independent. Namely, sacrifize 1 bit in $\ell$, and always apply $f_s(1, \cdot)$ when constructing the hash function $h_t(\cdot)$, and use $f_s(0, h_t(\cdot))$ on the outer layer. Using the "random function paradigm", $f_s(0, \cdot)$ and $f_s(1, \cdot)$ indeed look like two independent random function. In fact, in specific cases will not even have to do that (see below), even though it is a very inexpensive "loss" anyway. Below, we describe the hash function without the domain separation "trick" above.

To summarize, the advantage of using a PRF in bulding $\mathcal{H}$ is saving on the key size + making the construction possibly very efficient (since "practical" PRF's are very cheap). As a downside, the error probabilities will be worse, and will depend on the "computational

closeness" of our PRF to a truly random function. Namely, to prove the universality of the hash function, we first assume that $f_t$ is a truly random function (by the "random function paradigm"), and then prove the information-theoretic security as before.[1]

In all the examples below, we assume that: $m = m_1 \dots m_n$, where all $|m_i| = \ell'$, $L = \ell' n$, $\ell' \approx \ell$ (see below for details), *the number of blocks $n$ is fixed*,[2] and $t$ is a random key for our "base" PRF.

**Using XOR Mode.** Define

$$h_t(m_1 \dots m_n) = f_t(m_1, 1) \oplus f_t(m_2, 2) \oplus \cdots \oplus f_t(m_n, n)$$

(so that the input to the PRF is slightly longer: $\ell = \ell' + \log n$ bits long). Assuming $f$ is a truly random function from $\ell$ to $\ell$ bits, and if $(u_1, \dots, u_n) \neq (m_1, \dots, m_n)$, say $m_i \neq u_i$, we get that

$$\begin{aligned} \Pr_f[f(m_1, 1) \oplus \cdots \oplus f(m_n, n) = f(u_1, 1) \oplus \cdots \oplus f(u_n, n)] &= \Pr_f[f(m_i, i) \oplus f(u_i, i) = \alpha] \\ &= \frac{1}{2^\ell} \end{aligned}$$

where $\alpha$ is some string independent[3] of $f(m_i, i) \oplus f(u_i, i)$, which in turn is random since $u_i \neq m_i$. As we indicated, to build a PRF out of it, we actually use

$$f_s(0,\ f_s(1, m_1, 1) \oplus \cdots \oplus f_s(1, m_n, n))$$

**Using CBC Mode (CBC-MAC).** We can view this construction as simply applying the CBC mode of operation with IV being $0^\ell$ (a string of $\ell$ zeros), and outputting the *last block only* (remember, we do not need to "decrypt", only to "tag"):

$$h_t(m_1 \dots m_n) = f_t(m_n \oplus f_t(m_{n-1} \oplus \dots \oplus f_t(m_2 \oplus f_t(m_1)) \dots)) \tag{1}$$

The proof of (computational) universality of this $\mathcal{H}$ is a bit tricky, so we omit it. The main ideas are similar to what we have done earlier with CBC-encryption: intuitively, if $m \neq u$, say $m_i \neq u_i$, and $f$ is a truly random function, the values $h_t(m)$ and $h_t(u)$ "diverge once and for all" w.h.p., starting at the $i$-th application of the $f$.

**Lemma 1** *The function $h_t$ defined in Equation (1) is computationally AU.*

In order to get a PRF out of this variant of CBC, it seems like we need to apply an indepependent PRF $f_s$ to the $h_t$ above. Indeed, this variant is called *encrypted CBC-MAC*, and we will again come back to it in Section 4:

$$\text{Encrypted-CBC}(m) = f_s(f_t(m_n \oplus f_t(m_{n-1} \oplus f_t(\dots f_t(m_2 \oplus f_t(m_1)) \dots))))$$

[1] Of course, the construction will be inefficient with a truly random function, but this does not concern us: the efficient PRF construction is what we are using, only the *proof* uses a random function.

[2] See Section 4 for more on this restrictive assumption.

[3] That is why we used the block number inside $f$.

However, by revisiting the analysis of Lemma 1 more carefully, we see that it actually shows more. Namely, even without applying an outside $f_s$ to the above construction, we already get a PRF! More specifically, consider the following function known as the *CBC-MAC*, which is the same as the the function in Equation (1), except we renamed $t$ to $s$:

$$\text{CBC-MAC}(m) = f_s(m_n \oplus f_s(m_{n-1} \oplus f_s(\ldots f_s(m_2 \oplus f_s(m_1))\ldots)))$$

**Theorem 1** *CBC-MAC is a* PRF *on $L$ bit inputs, if $f_s$ is a* PRF *on $\ell$-bit inputs.*

The proof of this result is slightly tedious, but follows the same structure as the proof of almost universality we mentioned. Essentially, on any two distinct messages $m \neq u$, at the first message block $i$ where $m_i \neq u_i$, the current computation values of CBC-MAC($m$) and CBC-MAC($u$) will diverge *to random* once and for all. So all the output values are random and unrelated, meaning that we get a PRF.

CBC-MAC scheme is extremely popular, and is extensively used in practice. We also remark that we do not actually need $\mathcal{F}$ be a PRP family here (unlike for the encryption where we need to recover the message), any length-preserving PRF family is enough!

**Using Cascade Mode (and HMAC).** This next example builds a hash function $h_t : \{0,1\}^L \to \{0,1\}^\ell$ using a different PRF family $\{f_t\}$. Specifically, we do not care as much about the input size of $f_t$ (but the larger the better), let use call it $b$, but care that the output size is $\ell$ and the key size $k$ is at most $\ell$. In practice, for example, one uses input of size $b = 512$, and output and key size both either $\ell = 128$ or $\ell = 160$. However, we will see that the construction works even for $b = 1$!

Now, split the message $m$ into $m_1 \ldots m_n$, except now each chuck is of size $b$, so that $L = bn$. The initial key $t$ to $h_t$ is chosen at random from $\{0,1\}^\ell$, and then we inductively define values $x_0 \ldots x_n \in \{0,1\}^\ell$ as follows:

$$\begin{aligned} x_0 &= t \\ x_i &= f_{x_{i-1}}(m_i) \end{aligned}$$

Finally, the output $h_t(m_1 \ldots m_n) = x_n$. To describe it differently, $f_t(m_1)$ determines the PRF key $x_1$ to be used in the next round with input $m_2$, which in turn defines the PRF key $x_2$ to be used with the next input block $m_3$, and so on. This construction is called *cascade* or *Merkle-Damgard*. Notice, it really works for any input size $b \geq 1$, at the price of using $L/b$ evaluations of the underlying PRF $f$ (so larger $b$ yields more efificency).

The intuition behind this construction is quite similar to the case of CBC-MAC, and is the following. First, since all $x_i$'s are PRF outputs, they are computationally indistinguishable from random. Second, the very first block $i$ separating two $L$-bit messages $m$ and $u$ would result in two computationally independent PRF keys $x_i$ derived after the $i$-th call to $f$, and from this point on evaluating $h$ on $m$ and $u$ looks totally independent. Of course, with small probability the "chains" might "converge" again, but by simple birthday argument this convergence is quite unlikely (we omit formal bounds here). In particular, we can argue

**Lemma 2** *The cascade construction defines a computationally AU family of hash functions $\{h_t : \{0,1\}^L \to \{0,1\}^\ell\}$.*

In fact, the analysis shows more. Not only is $\mathcal{H}$ computational AU (meaning it can be composed with a *freshly keyed* PRF), but it is a PRF *by itself*! Intuitively, the above argument really said that the moment message diverge, everything stays random, and since any non-equal messages must diverge evetually, we get a PRF!

**Theorem 2** *The cascade construction defines a* PRF *from $L$ bits to $\ell$ bits.*

Why do we then care about Lemma 2 if we have Theorem 2? The reason will be clear in Section 4: it will have to do with our assumption that the message length $L$ is fixed, which is a bit too restrictive in practice. But now let us try to see what the cascade construction gives us:

(1) It actually gives us a PRF by itself. In fact, it turns any PRF with large enough output size (and not too large key size) into an arbitrary (but FIXED) length PRF, no matter how small the original input size $b$ is. In fact, when $b = 1$ the base PRF $f_t : \{0,1\} \to \{0,1\}^\ell$, where $|t| = \ell$, simply becomes a length doubling PRG $G : \{0,1\}^\ell \to \{0,1\}^{2\ell}$ via $G(t) = f_t(0) \circ f_t(1)$! Moreover, applying the cascade to this PRG $G$ reduces the cascade construction to the GGM construction of PRFs from PRGs! (check it yourself!) In essense, using larger $b > 1$ lets us use a $2^b$-ary tree instead of the binary tree, which brings the depth from $L = L/1$ to $L/b$ (meaning that one need $L/b$ evaluations of $f$ to compute the cascade).

(2) It also gives us a computational AU family of functions. Thus, if we combine it with another PRF $g_s : \{0,1\}^\ell \to \{0,1\}^c$, we get a composed PRF as well. The only problem here is that we would like implement $g_s$ using $f_s$, but the domains do not exactly match. $g_s$ should take $\ell$-bit inputs, and $f_s$ takes $b$-bit inputs (and outputs $\ell$-bit output). If $b \geq \ell$, which is the case in practice, this is not a problem: simply view the $\ell$-bit input $h_t(m)$ to $f_s$ as a $b$-bit input (i.e., pad it with $b - \ell$ zeros or something). Even otherwise, we can use Lemma 2 and build an $\ell$-bit input PRF $g_s$ out of $f_s$. But since this is never used, we'll assume $b \geq \ell$, and write $f_s$ to mean an $\ell$-bit input PRF (even though it can take potentially longer inputs). The resulting construction is called *NMAC*. More specifically, *NMAC* uses PRF $f$ from $b$ bits to $\ell$ bits (and key size $\ell$), where in practice $b \geq \ell$, has two independent keys $s$ and $t$, and essentially does $f_s(h_t(m))$, where $h_t$ is the cascades mode applied to $m$. In practice, we do not like to have two keys though, so a variant of *NMAC* which uses only one key is called *HMAC*. A sound implementation of *HMAC* should have sacrificed one input bit and prepended 0 for $s$ and 1 for $t$ like we described before, but instead it does something more heuristic. More or less, it sets $s = t +$ constant, where the constant is heuristically chosen and fixed. Thus, in the future we will only concentrate on the theoretically-cound *NMAC* mode.

# 3 A Different XOR-MAC

We also mentioning another popular MAC paradigm which uses a PRF's and the XOR mode of operation. Namely, let $\mathcal{F}$ be the PRF family and $\mathcal{H}$ be a hash family from $L$ to $\ell$ bits, whose properties will be given in a second. Rather than making the MAC output $f_s(h_t(m))$,

we now let it output $(nonce, f_s(nonce) \oplus h_t(m))$. The verification of $(nonce, v)$ checks that $v = f_s(nonce) \oplus h_t(m)$. Here *nonce* is the value that w.h.p. never repeats again, like a random string, or a counter (notice the similarity with encryption). In particular, this method is typically either randomized (*nonce* is random), or stateful (*nonce* is a counter), unlike our previous fully deterministic methods. Also, one has to either know or transmit the *nonce*. Finally, it is used only to make a MAC, and not a (more general) "long-input" PRF.

Still, what are the properties of $\mathcal{H}$ that make this method go through? As a simple attack, given a valid tag $(nonce, v)$ of $m$ and a value $a$, the adversary can try to output a "forgery" $(nonce, v \oplus a)$ for some $m' \neq m$. It is easy to see that this will be successful if and only if $h_t(m) \oplus h_t(m') = a$. Since $a, m, m'$ are arbitrary, at the very least we must have that for any $m \neq m'$, and any $a \in \{0,1\}^\ell$, we have

$$\Pr_t(h_t(m) \oplus h_t(m') = a) \leq \varepsilon$$

(where $\varepsilon$ is negligible). Such families are called *$\varepsilon$-xor-universal* (or *almost XOR-universal*, or simply, *AXU*). Notice, regular $\varepsilon$-universality corresponds to $a = 0$ since $h_t(m) = h_t(m')$ iff $h_t(m) \oplus h_t(m') = 0$. Thus, a further disadvantage of this method is that it uses more restrictive classes of hash functions! However, the latter criticism is typically not a big deal, since most natural universal families are actually xor-universal. It turns out that xor-universality is sufficient:

**Theorem 3** *$f_s(nonce) \oplus h_t(\cdot)$ defines a secure* MAC *whenever all the nonces are unique w.h.p., $\mathcal{F}$ is a* PRF *family and $\mathcal{H}$ is AXU.*

The most used xor-universal family comes from the XOR mode of the previous section (and uses PRF to build $h_t$):

$$h_t(m_1 \dots m_n) = f_t(m_1, 1) \oplus f_t(m_2, 2) \oplus \cdots \oplus f_t(m_n, n)$$

It is easy to see that our proof from the previous section in fact showed that $\mathcal{H}$ is AXU (check it). As in the previous section, we have to use the trick with prepending 0 and 1 to make the final MAC construction and use the same key:

$$\mathsf{Tag}_s(m) = (nonce,\ f_s(0, nonce) \oplus f_s(1, m_1, 1) \oplus \cdots \oplus f_s(1, m_n, n))$$

This is called the XOR-MAC. Naturally, it has a randomized or counter flavor depending on whether the *nonce* is random, or is a counter (in the later case the *nonce* need not be explicitly sent over).

But why use this method given its two disadvantages (only a MAC + slightly stronger assumption of $h$)? The point is that the security of $\mathcal{H}$ depends on the *output size* of the PRF, rather than the *input size* like we had in the $f_s(h_t(m))$ composition. Namely, if previously $h : \{0,1\}^L \to \{0,1\}^{\underline{\text{input length of}}\, f}$, now we have $h : \{0,1\}^L \to \{0,1\}^{\underline{\text{output length of}}\, f}$. And since it is much easier to extend the output of a PRF than it's input, we get that this XOR-MODE might yeild considerably better exact security in practice, especially if used with counters (so that one does not have to pay a birthday bound on *nonce* which depends on the input length of $f$). Overall, which MAC is better depends on a variety of parameters, with most constructions being incomporable (i.e., for different circumstances either one could be better).

# 4 Variable Length-Inputs

So far we made a convenient simplifying assumption that all the inputs to a MAC are of the same length $L$. In practice, this assumption is extremely inconvenient, and we would like to build *variable-length* MACs: namely, MACs which work for any input size in $\{0,1\}^*$.

First, let us revisit our constructions so far (whose key length is independent of the message length), and see which ones are right away secure variable-length MACs. As we will see, the answer is essentially "all except cascade and CBC-MAC".

- **Polynomial Construction.** Although the construction is insecure the way we stated it, — since the polynomial corresponding to $m_1 \dots m_n$ is the same as the one corresponding $0^\ell m_1 \dots m_n$, — it is very easy to fix. Simply prepend a fixed non-zero block $a$ to each message. Thus, polynomial corresponding to $m$ is now $q_m(x) = ax^n + m_n x^{n-1} + \ldots + m_1$. Now if $m$ has $n$ blocks, $u$ has $b$ blocks, and $m \neq u$, then the difference between $q_m(x)$ and $q_u(x)$ is a *non-zero* polynomial of degree at most $\max(n, b)$. Indeed, if $n = b$, then $ax^n$ cancels, but the remaining polynomials won't cancel since $m \neq u$; else, say $n > b$, the term $ax^n$ will not cancel (and, similarly, when $n < b$).

- **AU-based or AXU-based XOR Modes.** It is easy to see from the analyses of either mode that it can directly handle variable-length messages, since the block number is always included when evaluating $f_t$, so both the computational AU and the AXU properties still hold.

- **CBC-MAC and cascade.** It is not hard to see that either one of these modes is *not secure* when dealing with variable length messages. We give the reason for the cascade, leaving the (slightly more complicated) attack on the CBC-MAC as an exercise. The problem is the so called *extension attack*. Given a cascade of the message $m = m_1 \dots m_n$, we can easily forge a tag for any extended messages $m' = m_1 \dots m_n m_{n+1} \dots m_b$, where $b > n$. The reason is that the output of $x = cascade(m)$ is the PRF key we need to plug in to continue evaluating $x' = cascade(m')$ starting from the $(n+1)$-st block. Specifically, $x'$ is simply the cascade of $m_{n+1} \dots m_b$ with the key $x$. Thus, if we learn $x$, we can compute the tag of any extended message by ourselves! Thus, Theorem 2 and Theorem 1 are not true for variable-length messages!

  On the positive side, it is easy to see that the above attack if the *only* attack on the cascade and the CBC-MAC. In particular, if we encode messages we tag in a *prefix-free* form, — namely, no encoded message is a prefix of another encoded message, — the cascade and the CBC-MAC are still secure.

- **Encrypted CBC-MAC and HMAC.** We claim that the encrypted versions of CBC-MAC and cascade (i.e., the NMAC) are still secure, even for variable-legth messages. To prove this, we only need to show that CBC-MAC and cascade remain *computational almost universal* even for variable length messages. In other words, we claim that Lemma 1 and Lemma 2 are still true!

  The argument is an extension of the one used to prove Lemma 1 and Lemma 2. There, we used the fact that ones the messages $m \neq u$ "diverge", they never meet again. As

we said, this analysis also works for prefix-free messages $m \neq u$. In the general case, say, when $m$ is a prefix of $u$, we also have to argue that it is unlikely to have short "cycles". The argument is not very hard, and uses the same kind of birthday bounds we gave so far. So we will omit it from here.

To summarize, for variable-length messages, it is always safe to use HMAC and encrypted CBC-MAC. If one additionally knows (or can enforce) the messages to be prefix-free, then basic cascade and CBC-MAC are also secure.

# 5 CCA-secure and Authenticated Encryption

We briefly touch upon more advanced topics. First, recall from the homework the notion of CCA-security.

DEFINITION 1 SKE $(\mathsf{Gen}, E, D)$ is IND-secure against CCA attack iff $\forall \mathsf{PPT} B = (B_1, B_2)$,

$$\Pr[\, b = \tilde{b} \;\mid\; \begin{array}{l} s \leftarrow G(1^k); \\ (m_0, m_1, \beta) \leftarrow B_1^{E_s, D_s}(1^k); \\ b \leftarrow \{0,1\}; \\ \tilde{c} \leftarrow E_s(m_b); \\ \tilde{b} \leftarrow B_2^{E_s, D_s}(\tilde{c}, \beta); \,] \leq \frac{1}{2} + \mathsf{negl}(k) \end{array}$$

where $B_2$ cannot call $D_s$ on input $c$. ♢

The definition is quite natural, and allowes the attacker to have oracle access to both the encryption and decryption functionality. Recall also from the homework that the following encryption scheme, based of a pseudorandom permutation $g_s$ is CCA-secure: $E_s(m; r) = g_s(m \circ r)$, where $m \circ r$ is concatentation of $m$ and randomness $r$. The decryption simply recovers $m \circ r$ by computing $g_s^{-1}(c)$ and "drops" $r$.

We will now give another way of constructing CCA-secure schemes, which is more general and has stronger security properties. In fact, recall that encryption schemes and message authentication *schemes* have roughly the same syntax, but different goals. Both take message $m$, and convert it into some other "enveloped" message $c$, by using the secret key $s$. And the recipient should recover $m$ (or output invalid) from $c$, again using $s$. For encryption, we cared about privacy: no information about $m$ should be contained in $c$, while for authentication we cared about authenticity: the recipeint should be sure that $m$ came from the sender, irrespective of whether or not $m$ is hidden. What if we combine these two goal? We get an extremely useful primitive called authenticated encryption.

In brief, authenticated encryption is again a triple of algorithms $(G, E, D)$. $G$ is the key generation algorithm, i.e. $G(1^k)$ produces the shared secret key (usually, a truly random sting of some length). As usual, $c \leftarrow E_s(m)$ produces the ciphertext, while $D_s(c) \rightarrow \tilde{m} \in M \cup \{\perp\}$, where $M$ is the message space (say, $M = \{0,1\}^k$), and $\perp$ denotes "invalid". For privacy, we want $(G, E, D)$ to be an IND-secure encryption scheme against CPA (chosen plaintext attack). However, now we also want $(G, E, D)$ to be a secure message authentication scheme (strongly) existentially unforgeable against chosen message attack. Notice, here a successful forgery constitutes producing $c$ s.t. $D_s(c) \neq \perp$, and $c$ was

never returned by the "tagging" (i.e., "encryption")[4] oracles. Also notice the attacker does not necessarily need to "know" the message he is forging.

It is a good exercise (see homework) to see what is "wrong" with the CCA-secure encryption scheme $g_s(m \circ r)$ mentioned above, in the context of authenticated encryption. In fact, in the homework you will show that $(r, g_s(m \circ r))$ is in fact a secure authenticated encryption. Here, however, we want to mention several useful properties of authenticated encryption:

**Theorem 4** *A secure symmetric-key authenticated encryption is always a* CCA*-secure symmetric-key encryption. Namely,* CPA*-security coupled with (strong) unforgeability implies* CCA*-security.*

The theorem is simple, so we only hint on the proof. Intuitively, if the scheme is unforgeable, then the decryption oracel is "useless": either the attacker already knows the answer (i.e., he got it from the encryption oracle before), or he gets $\perp$ (which is "useless") or he forged a valid ciphertext never returned by the encryption oracle.

Thus, instead of directly shoing CCA-security, it suffices to show CPA-security plus unforgeability. In fact, doing so will give a strictly stronger primitive of authenticated encryption!

**Constructions of AE.** This is an active area of research, and there are many interesting constructions. We already mentioned one above. Here we just metnion another: encrypt-then-mac. The secret key consists of two keys $s$ for CPA-secure encryption and $u$ for strongly unforgeable MAC. Then, $AE_{s,u}(m) = (c \leftarrow Enc_s(m), t \leftarrow Tag_u(c))$, while authenticated decryption first checks if $t$ is a valid tag of $c$, and only then outputs $m = Dec_s(c)$. We leave it a simple exercise to argue the security of this scheme.

[4]Notice, in this scenario the "tagging" and "encryption" oracles are *the same*.

| CSCI-GA.3210-001<br>MATH-GA.2170-001 | **Introduction to Cryptography** | April 11, 2012 |
|---|---|---|

Lecture 12

*Lecturer: Yevgeniy Dodis* *Spring 2012*

We saw that $\varepsilon$-universal hash families are very useful in the design of pseudorandom functions (and, thus, message authenticated codes). We will see, however, that we will need stronger hash functions when we design digital signature schemes. Such hash functions are called *collision-resistant* and this lecture is primarily dedicated to their definition and constructions. Interestingly, we will see that CRHFs are also useful in the domain extension of MACs which are not necessarily PRFs.

At the end nof the lecture we will introduce digital signature schemes, and show how CRHFs simplify the design of such signatures via the "hash-then-sign" paradigm. The lecture will conclude with some advanced topics which are completely optional to study.

# 1 Motivation: Domain Extension of MACs

Recall, if $g_s$ is a PRF, and $h_t$ is $\varepsilon$-universal (for negligible $\varepsilon$), then $f_{s,t}(m) = g_s(h_t(m))$ is a PRF as well. What if $g_s$ is "only" a MAC, and not necessarily a PRF? The proof we had before breaks down. In fact, one can construct and artificial MAC $g_s$ which is *not* a PRF, for which the function $f_{s,t}$ is not secure even with *natural* $\varepsilon$-universal functions, like matrix-vector multiplication. Thus, if we hope for the "hash-then-mac" approach to work much like "hash-then-prf" did, we need a stronger kind of hash function $h_t$.

It turns out that such strong type of $h_t$ exists, and is they are called *weakly collision-resistant* hash functions (WCRHFs). Intuitively, a WCRHF $h_t$ has the property that the attacker cannot find two distinct inputs $x \neq y$ such that $h_t(x) = h_t(y)$, even *when having oracle access to* $h_t(\cdot)$ (recall, $t$ is a secret key). In other words, for $\varepsilon$-universal hash functions the attacker had to find $x$ and $y$ *without* oracle access to $h_t$, while here we allow such access. We will not prove it (it's not very hard), but the following the the corresponding theorem for the domain extension of MACs:

**Theorem 1** *If $g_s$ is an $\ell$-bit* MAC*, and $h_t$ is a* WCRHF *from $L$ bits to $\ell$ bits, then $f_{s,t}(m) = g_s(h_t(m))$ is an $L$-bit* MAC*.*

Intuitively, the attacker $A$ forging the MAC $f_{s,t}$ and producing a forgery $m$ either had to use $m$ such that $h_t(m)$ is "new" (different from $h_t(m_i)$, where $m_i$ are the message MAC'ed by $A$), or $h_t(m) = h_t(m_i)$, fior some $i$ and $m \neq m_i$. The first case easily leads to the forgery of the MAC $g_s$ (on message $h_t(m)$), while the second case leads to a collision $(m, m_i)$ for $h_t$.

Unfortunately, most $\varepsilon$-universal hash functions are not weakly collision-resistant (in fact, WCRHFs imply one-way functions, but this is tricky to show). Still, it is possible to construct relatively simple WCRHFs (simpler than PRFs), but this topic is too advanced for this class. Instead, we notice that the domain extension of MACs would be even simpler if we had an even stronger type of function: a (strongly) *collision-resistant* hash function

(CRHF). Intuitively, $h_t$ where collisions are hard to find *even when giving the key $t$ to the attacker*! We define such functions in the next section.

## 2 Collision-Resistant Hash Functions

Let $k$ be the security parameter, and $\mathsf{Gen}(1^k)$ be the generation algorithm which outputs a public key $PK$, and, if needed, a description of the message space $\mathcal{M} = \mathcal{M}(PK)$ out output space $\mathcal{R} = \mathcal{R}(PK)$. We require that $|\mathcal{M}| > |\mathcal{R}|$. In fact, for simplicity, below we assume $\mathcal{M}(k) = \{0,1\}^{L(k)}$, $\mathcal{R}(k) = \{0,1\}^{\ell(k)}$, where $L(k) > \ell(k)$. However, all the discussion easily generalizes to any domain $\mathcal{M}$ and range $\mathcal{R}$, as long as $|\mathcal{M}| > |\mathcal{R}|$. In particular, as we explain later, in practice $\mathcal{M}$ will be equal to $\{0,1\}^*$ (all finite strings) rather than $\{0,1\}^L$.

Definition 1 A family of functions $\mathcal{H} = \{h_{PK} : \mathcal{M}(PK) \to \mathcal{R}(PK)\}$ generated by $\mathsf{Gen}$ is called a family of *collision-resistant hash functions* (CRHFs) if (a) $|\mathcal{M}| > |\mathcal{R}|$, (b) $h_{PK}$ is efficiently computable for any $PK$, and (c) for any PPT attacker $A$,

$$\Pr[x \neq x' \ \wedge \ h_{PK}(x) = h_{PK}(x') \mid PK \leftarrow \mathsf{Gen}(1^k), (x,x') \leftarrow A(PK, 1^k)] \leq \mathsf{negl}(k)$$

$\diamondsuit$

To compare with $\varepsilon$-universal hash functions, there the description of the function is chosen at random *after* the attacker committed to $x \neq x'$. In contrast, here the attacker learns the description $PK$ of the CRHF before trying to find a collision. In particular, we had information-theoretic $\varepsilon$-universal function families, while any family of CRHFs must be based on *computational* assumptions. Indeed, since $|\mathcal{M}| > |\mathcal{R}|$ for any public key $PK$, any $PK$ has some non-trivial (i.e., $x \neq x'$) collisions, so the only reason $A$ cannot find them is because $A$ is computationally bounded.

Application to MACs. Notice, Theorem 1 is really trivial to see with CRHFs in place of WCRHFs. We leave it an an exercise, later proving an anlogous results with digital signatures.

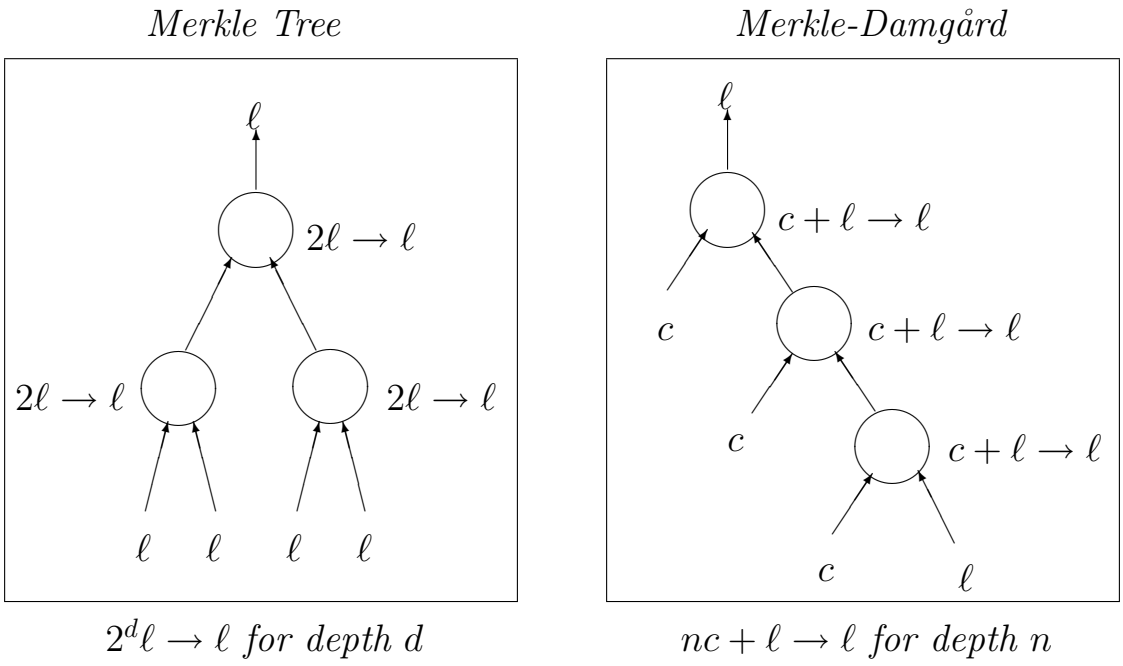
## 3 Extending the Domain of CRHF

Before constructing specific CRHFs, we study the question of their domain extension.

General Composition Idea. Here is a general idea. If it is too general, you can skip this paragraph and look at our two main examples. Let $T$ be any tree where each internal nodes of this tree has an *ordered* list of its chidren. A valid labeling of this tree corresponds to: (1) associating some string with each leaf of the tree; (2) associating some function with every internal node of the tree; (3) making sure the associations above are *consistent* in the following sense. Starting with the leaves and moving up towards the root, we require for each internal node $N$ the following: if $N$ is associated with a function $f$ from $b_1$ to $b_2$ bits and has children $N_1 \ldots N_t$, then sum of lengths of strings $s_1 \ldots s_t$ associated with $N_1 \ldots N_t$ is exactly $b_1$. If this is true, we associate the string $s = f(s_1, \ldots, s_t)$ with $N$ and move up the tree (thus, the string associated with $N$ has length $b_2$). At the end, the label of the whole tree is the label of its root. Essentially, we want to ensure that "all the lengths match".

In our compositions, we start with some hash family $\mathcal{H}$, and construct a new hash family $\mathcal{H}'$ with much larger input length, by means of some conveniently chosen tree $T$. The functions we put in the internal nodes of $T$ will be the hash functions from the corresponding original family $\mathcal{H}$, the labels of the leaves will be parts of the much longer input to the functions in $\mathcal{H}'$, while the label of the tree (the root) is the output of the new hash fucntion.

Merkle-Damgård and Merkle Tree. Two main example we will use are given below. In both examples, one chooses a single hash function $h$ from $\mathcal{H}$ at random and uses it at all levels of the tree. However, for the sake of generality and because it will be useful a bit later, we denote the hash function associated with internal node $i$ by $h_i$, despite the fact that for our current purposes all of them are really equal to the same $h$.




- **Cascade (or Merkle-Damgård).** Assume $\mathcal{H}$ goes from $\ell + c$ to $\ell$ bits for some $c > 0$ (in fact, $c$ could be as small as 1). Set $L = \ell n + c$, write input $x \in \{0,1\}^L$ as $x = x_0 \circ x_1 \circ \ldots \circ x_n$, where $|x_0| = \ell$ and each $|x_i| = c$ for $i > 0$. Given functions $h_1 \ldots h_n \in \mathcal{H}$, set

$$H(x_0 \circ x_1 \circ \ldots \circ x_n) = h_n(x_n \circ h_{n-1}(x_{n-1} \circ \ldots \circ h_2(x_2 \circ h_1(x_1 \circ x_0)) \ldots))$$

  Notice that this function family corresponds to a tree which is a long path of depth $n$. A useful sub-example corresponds to $c = 1$ and $n = \ell - 1$. Then we get $H$ going from $2\ell$ to $\ell$ bits from $h_i$'s going from $\ell + 1$ to $\ell$ bits each.

  The Merkle-Damgård transform is extensively used in practice. Although it uses a tree which is just a long path, it is very convenient since: (a) it works for any value of $c > 0$, making it very flexible; (b) it allows to process the data in one pass, from left to right (in particular, one does not need to know the length $L$ before one starts); and (c) the lack of parallelism (compared with the second example below) is usually not

very important in practice, since the data often comes in sequentially and/or parallel machines are anyway not readily available.

- **Merkle Tree.** A more interesting example comes from the complete binary tree (CBT), when we start from the family from $2\ell$ to $\ell$ bits. Using a complete binary tree of depth $d$, we get a new family from $2^d\ell$ to $\ell$ bits.

  The most obvious advantage of the Merkle tree isea is that it allows to process data in parallel, as opposed to the Merkle-Damgård transform which is sequential. However, we already remarked that this is not very important in practice. Instead, the main reason Merkle Tress are extensively used in practice in the following. Assume you have a big file $F$ consisting of $2^d$ blocks, and you wish to "commit" to its contexts by publishing a collision-resistant hash of the file. For example, you want to store $F$ on an untrusted server, but remember the short hash value $y = H(F)$ of $F$. Now, you want to retrieve the $i$-block $F_i$ of $F$ from the server. The server can just send you $F_i$, but you do not trust the server. Alternatively, the server can send you the entire file $F$, and you verify that the hash of $F$ is $y$. Here the server cannot cheat, but the protocol is very wasteful. You only wanted one short block $F_i$, but had to read all $2^d$ blocks, and recompute the entire hash function on all of $F$.

  Merkle trees allows you to do much better. As before, you only store the hash $y$, which is the root of the Merkle tree. You also let the server store the entire Merkle tree (i.e., the hash values on all the internal nodes, which at most doubles the storage on the server's side). Now, the server can only send you: (a) the block $F_i$ you want; (b) all the internal hash values $y_d = F_i, y_{d-1}, \ldots, y_1, y_0 = y$ on a path from $y$ to $F_i$; (c) all the "sibling" nodes corresponding to $y_{d-1}, \ldots, y_0$ (i.e., make sure the client gets back both children of $y_0, \ldots, y_{d-1}$; one child is anyway present because of (b), but sending the sibling ensures that the second child is present too). Thus, the server sends a total of $2d + 1$ blocks, which is dramtically less than $2^d$. Now, the client just evaluates little $h$ for each of the $d$ internal nodes on the path from $F_i$ to $y$, and if all the checks are valid, the client knows $F_i$ is correct. This means only $d$ evaluations and only $2d + 1$ blocks, which is logarithmic in the length $2^d$ of the file.

  To summarize, Merkle trees is extremely useful if one wants to reliably retrieve individual message blocks from some untrusted storage, without the necessity to read the entire file.

Global Collision Implies Local Collision. The main structural property of any tree (including the above examples) is the following.

**Lemma 1** *Assume $H(x) = H(x')$ for some distinct $x, x' \in \{0,1\}^L$. Then there is (easy to find) internal node $i$ of the tree (labelled by some $h_i \in \mathcal{H}$) such that: (1) it takes input $x_i \in \{0,1\}^\ell$ when evaluating $H(x)$ and input $x'_i \in \{0,1\}^\ell$ when evaluating $H(x')$; (2) $x_i \neq x'_i$; (3) $h_i(x_i) = h_i(x'_i)$. In other words, a "global collision" in $H$ implies there exists a "local collision" in at least one of the $h_i$'s.*

**Proof:** Imagine a moving "frontline", described below, and let's see how this frontline evolves in the "world" $W$ of $x$ vs. the "world" $W'$ of $x'$. Initially, the fronline consists of all

the (labels of the) leaves of the tree, so that $F = x$ in $W$ and $F' = x'$ in $W'$. By assumption $F \neq F'$ originally. At each next step, we take one internal node, adjacent to the frontline (all its children are in the frontline), include it (or rather its label) in the frontline, but remove its children (or rather, their labels) from the frontline. We do it until we reach the fronline which just contains the root. Now, the final fronline is equal to the value of $h'$ (the label of the root). Sicne $H(x) = H(x')$, the final values of the fronline are the same in $W$ and $W'$. Thus, the fronlines were different at the beginning in $W$ and $W'$, but became the same at the end. Hence, the must be an intermediate step, where we included some node $i$ in the fronline (with function $h_i$) so that: $F \neq F'$ before including $i$, but $F' = F$ after including $i$. But this literally means that the inputs $x_i$ and $x'_i$ to $h_i$ were different (since this is the only way for the frontline to differ originally), but the outputs of $h_i$ are the same. This exactly means we found the local collision requested. □

Concrete Examples. As we already mentioned, for our purposes we will use the same randomly selected function $h \in \mathcal{H}$ at each node of the tree, no matter how the tree looks like. Indeed, assuming $H(x) = H(x')$ for some distinct $x, x' \in \{0,1\}^L$, Lemma 1 implies that there exists some (easy to find) distinct $x_i, x'_i \in \{0,1\}^\ell$ so that $h(x) = h(x')$. Thus, if some $A$ can fund a collision in $H$ (build using the same $h \in \mathcal{H}$), one can use $A$ to find a collision in $h$ itself. It turns out we get the same paramenters irrespective of which "legal" tree we are using. For convenience, below we apply this composition to the "path", the "complete binary tree" ("CBT"), and their combination (first get length doubling by path, then apply CBT).

**Theorem 2** *Let $\mathcal{H}$ be a* CRHF *family of fucntions from $\ell + c$ to $\ell$ bits and key length $p$. Then there exists the following* CRHF *$\mathcal{H}'$ going from $L$ to $\ell$ bits, and using the same key $h$ of length $p$, where the evaluationg of $H \in \mathcal{H}'$ takes $\frac{L-\ell}{c}$ evaluations of $h$. For concrete examples,*

- *By using the Merkle-Damgård chaining mode, we can achieve $L = nc + \ell$, where evaluation of $H \in \mathcal{H}'$ takes $n = \frac{L-\ell}{c}$ evaluations of $h \in \mathcal{H}$. In paricular, if $c = 1$ we can get $L = 2\ell$ using $\ell$ evaluations of $h$.*
- *If $c = \ell$ (i.e., $h$ goes from $2\ell$ to $\ell$ bits), using the Merkle tree of depth $d$ we can achieve $L = 2^d\ell$, where evaluation of $H \in \mathcal{H}'$ takes $(2^d - 1) = \frac{L}{\ell} - 1$ evaluations of $h \in \mathcal{H}$.*
- *Combining the above, if $c \leq \ell$, we can achieve $L = 2^d\ell$, where evaluation of $H \in \mathcal{H}'$ takes $\frac{\ell(2^d-1)}{c} = \frac{L-\ell}{c}$ evaluations of $h \in \mathcal{H}$. In particular, when $c = 1$ we use $(L - \ell)$ evaluations of $h : \{0,1\}^{\ell+1} \to \{0,1\}^\ell$.*

**Corollary 2** *Given a fixed* CRHF *$\mathcal{H}$, we can make a new* CRHF *$\mathcal{H}'$ with essentially unbounded input size, without increasing the key size for $\mathcal{H}'$.*

Variable-Lengh Inputs. So far we considered the case of a fixed tree, corresponding to a fixed-length inputs (of length $L$). Just like for the case of MACs, in practice one want to de able to handle arbitrary-length messages; i.e., $\mathcal{M} = \{0,1\}^*$. We restrict our attention

to how to extend the Merkle-Damgård chaining to handle such inputs, since this is what is being used in practice.

Assume we have a collision-resistant compression function $h$ from $\ell + c$ to $\ell$ bits. We will assume that $c \geq \log L$, where $L$ is the *largest* length of the message being hashed (i.e., all messages are of length at most $2^c$). This is not a big restriction, since in practice $c \geq 128$ (usually 512), and $2^{128}$ is larger than the number of molecules in the universe. Finally, we will assume that input length (whatever it is) is a multiples of block size $c$: if not, uniquely pad the input to make its length a multiple of $c$. We set the initialization vector $IV$ to arbitrary $\ell$-bit constant (say, $0^\ell$), and consider first the usual cascade construction applied to $m = m_1 \dots m_n$, where $|m_i| = c$:

$$H(m_1 \dots m_n) = h(m_n, h(m_{n-1}, \dots h(m_1, IV), \dots))$$

We immediately notice that this Merkle-Damgård chaining preserves collision-resistance as long as the inputs are encoded in a *suffix-free* form. Namely, no legal input $m$ is a suffix of another input $m'$. Indeed, if this is not the case, then traversing any collision backward on any $m \neq m'$ eventually leads to distinct inputs colliding wr.t. $h$ in one of the blocks: namely, the first value $j \geq 0$ such that $m_{n-j} \neq m'_{n'-j}$, where $n$ and $n'$ are the number of blocks in $m$ and $m'$, respectively.

What if inputs are not in the suffix-free format? Well, it's easy to make them into such format. Simply add block $m_{n+1} = \langle n \rangle$: namely, add the number of message blocks as the last block of the encoded message. Clearly, if $n \neq n'$, then the last block is different, and we get suffix-freeness. If $n = n'$, the the last block is the same, but at least one of the message blocks is different since $m \neq m'$. This is called *Merkle-Damgård strengthening*.

$$H(m_1 \dots m_n) = h(\langle n \rangle, h(m_n, h(m_{n-1}, \dots h(m_1, IV), \dots)))$$

The nice thing about this method is that one does not have to know the length of the message $n$ before hashing it. Simply keep counter of how many blocks you hashed so far, and when you encounter the end of the file marker, you know $n$ and can apply the Merkle-Damgård strengthing at this point.

To summarize, when we construct CRHFs in the next section, it suffices just to get some level of compression, from which we can efficiently extend the domain to $\{0,1\}^*$ via Merkle-Damgård strengthening.

# 4 Constructions of CRHFs

It turns out, the existence of CRHFs is a powerful and strong assumption. In particular, we do not know how to build CRHFs from OWFs, OWPs, and even TDPs! In fact, there is some strong theoretical indication that it is impossible to build CRHFs from general TDPs. Luckily, we can construct CRHFs using:

- **Ideal Block Ciphers.** This is a *heuristic* construction (explained later in Section 4.1), but is extensively used in virtually all current hash functions such as SHA-1 or MD5. So we cover it because of its widespread use and the fact that we can give some partial theoretic justification.

- **Claw-Free Permutations.** This is a very strong assumption (studied later in Section 4.2), much stronger than TDPs, for example. Luckily, we can build simple claw-free permutations (CFPs) from all the number-theoretic assumptions we studied so far, including RSA, factoring and discrete log. In particular, this shows how to build CRHFs from all these number-theoretic assumptions.

- **Number-Theoretic Assumptions.** We already mentioned above that one can build CFPs, and thus, CRHFs, from all the number-theoretic assumptions we studied so far, including RSA, factoring and discrete log. However, in each of these cases one can build more efficient CRHFs but going into the specifics of number theory. This is briefly mentioned in Section 4.3.

To summarize, although the existence of CRHFs is a strong assumption, we can build them effiicently using a variety of techniques, which means that in the sequel we will freely use CRHFs.

## 4.1 Construction from Ideal Cipher

In practice, suprisingly enough, CRHFs are built from block ciphers. Let $E_s(x)$ denote a block cipher with key $x$ and input $x$, and $E_s^{-1}(y)$ denotes the corresponding inverse. We let $|x| = |y| = n$ and $|s| = k$. The following construction, called *Davies-Meyer constrcution*, is used quite extensively for most practical hash functions, including SHA and MD5 (the latter recently broken). It builds the following compression function $h : \{0,1\}^{n+k} \to \{0,1\}^n$:

$$h(s, x) = E_s(x) \oplus x$$

The claim is that the DM construction is a good CRHF "in practice". Namely, it is infeasible to find $(s, x) \neq (s', x')$ s.t. $E_s(x) \oplus x = E_{s'}(x') \oplus x'$. This claim might seem puzzling at first, and for a good reason. First, we said that it is unlikely we can construct CRHFs from OWFs, and we know (via Luby-Rackoff) and block ciphers can eventually be built from OWFs. More seriously, we see that the construction does something "un-kosher": the secret key $s$ for the block cipher is simply an input to $h$ *can be chosen by the attacker* trying to find collision! Indeed, it is easy to construct an artificial PRP for which the DM construction is insecure. I.e., assume $E_0(0) = 0$ and $E_1(1) = 1$. This does not contradict the PRP assumption, since it is very unlikely that a random key $s$ will be 0 or 1. Thus, indeed, we can never prove security assumpting that our block cipher $E$ is "only" a PRP.

Nevertheless, in practice people try not to chose block cipher with intentionally "weak" keys, like the artificial counter-example above. In fact, ideally, we hope that a good block-cipher should behave like a random permutation *for every key*. Unfortunately, this assumption is too strong, and it is hard to make it formal: once the code of the block cipher is fixed, it is *not* random for *every* key. In fact, for every particular key the permutation is fully determine. Still, in practice the cipher seems to be "good enough" to avoid any "silly counter-examples" like above.

Because of these considerations, theoreticians and practitioners converged on the following abstract model of block ciphers, called the *ideal cipher model* (ICM). The best way to imagine the ICM to to pretend that there exists a trusted third party Zak, who chose

in his head $2^k$ *truly random permutations* on $n$ bits, $E_s(\cdot)/E_s^{-1}(\cdot)$, for every $s \in \{0,1\}^k$. Now, whenever a party — either honest one (like Alice and Bob), or malicious (like Eve) — need to compute $E_s(x)$ or $E_s^{-1}(y)$, they simply ask Zak, and he gives them the answer. Notice, this makes perfect mathematical sense, and will allow us to prove theorems about the ICM. On the other hand, in practice there is no Zak, and people simply use a good enough block cipher, like AES. And the hope is that the block cipher is so good, that no party — even Eve — can do much more than honestly evaluate it on a bunch of points, and more or less assume that the result will be totally unpredictable. i.e., despite having the code of $E$, Eve cannot underastand the complexity of the code and can only run the primitive forward/backward, always expecting a random(-looking) result.

DM is Collision-Resistant in ICM. If this was too abstract, let us semi-formally argue the collision-resistance of the DM construction in the ICM.

**Theorem 3** *$h(s,x) = E_s(x) \oplus x$ is a collision-resistant hash function in the ICM.*

**Proof:** Assume there exists an attacker $A$, who expects oracle access to $E_\cdot(\cdot)$, $E_\cdot^{-1}(\cdot)$ (i.e., can chose both the key and the input/output), and find a collision $(s,x) \neq (s',x')$ for $h$ with probability $\varepsilon$. We will prove that $\varepsilon$ must be low, using an *information-theoretic argument* (this is possible because we are using ICM!). We do a sequence of games, and let $p_i$ be the probability the attacker still finds collision in Game $i$.

- *Game 0.* The orginal game between $A$ and Zak, who chose a totally random ideal cipher.
- *Game 1.* This is lazy Zak, but otherwise the same as non-lazy Zak. Namely, instead of choose the cipher in full at the beginning, he defines it incremetally, as requested by $A$. He keeps a table $T$ of tuples $(s,x,y,z)$, which correspond to defined values $E_s(x) = y$ (and $E_s^{-1}(y) = x$), where for conveneince Zak also stores $z = x \oplus y$. Initially $T$ is empty (nothing defined), but $T$ grows with each query of $A$. Namely, for forward query $(s,x)$, if a tuples $(s,x,y,z)$ is in the table, give back $z$. (Same for backways query $(s,y)$, returning $x$.) But if such tuples is not there, choose a random $y$ distinct from all $y'$ already defined for the same key $s$ (i.e., if $E_s(x') = y'$, then exclude $y'$ from consideration). Same for $E_s^{-1}$. Obviously, $p_1$ is still $\varepsilon$, since this is the same game.
- *Game 2.* Identical to Game 1 above, except Zak does not exclude any $y$ when defining a random forward query (or $x$ for backward query). If $A$ makes $q$ queries, for each such query Zak could only run in trouble (violating permutation structure) with probability at most $q/2^n$, so the total probability of him running into trouble after all queries is at most $q^2/2^n$. Thus, $|p_2 - p_1| \leq q^2/2^n$.
- *Game 3.* Identical to Game 2 above, except Zak checks if the new values $z_i$ that he puts in the table "for fun" ever collide. If so, he stops and loses the game. For each new forward query $(s,x)$ or backward query $(s,y)$ which defines a new $z$, we notice that $z$ is defined at random: either one chooses random $y$ and sets $z = x \oplus y$, or chooses random $y$ and again sets $z = x \oplus y$. Thus, the chance that any of the $z$'s repeat is simply a birthday bound $q^2/2^n$. Hence, $|p_3 - p_2| \leq q^2/2^n$.

- *Game 4.* Identical to Game 3 above, except when $A$ outputs a claimed collision $(s,x) \neq (s',x')$, we make sure that we already defined the values $E_s(x)$ and $E_{s'}(x')$. I.e., that $T$ contains tuples $(s,x,y,z)$ and $(s',x',y',z')$ for some $y,z,y',z'$. If this is not so, pretend that $A$ asks Zak (from Game 3) to evaluate these two more queries for him. Clearly, this is the same as Game 3, except Zak could "screw up" answering the last two queries, which happens with probability $4q/2^n$ (each query can mess up with probability at most $q/2^n$, by the analyses of Games 2 and 3). Hence $|p_4 - p_3| \leq 4q/2^n$.

But now, let us argue that $p_4 = 0$. Indeed, in Game 4 both inputs $(s,x)$, $(s',x')$ have two distinct entries in $T$, for which $z$ and $z'$ are distinct (by Game 3). Thus,

$$h(s,x) = E_s(x) \oplus x = x \oplus y = z \neq z' = x' \oplus y' = E_s(x') \oplus x' = h(s',x')$$

Tracking back, we see that $\varepsilon = p_0 \leq (2q^2 + 4q)/2^n$, which is negligible. □

We saw that the above ICM proof was information-theoretic, despite the fact that CRHFs imply OWFs. This is because ICM model is very strong and should be used with extreme caution!

### 4.2 Construction from Claw-Free Permutations

We now turn to constructions in the standard model. W estart with a general construction. It uses a notion of *claw-free permutations*. We define them below

DEFINITION 2 A family of functions $\mathcal{F} = \{(f_{0,PK}, f_{1,PK}) : \mathcal{M}(PK) \to \mathcal{M}(PK)\}$ generated by Gen is called a family of *claw-free permutations* (CFPs) if (a) $f_{0,PK}$, $f_{1,PK}$ are efficiently computable *permutations* for any $PK$; and (b) for any PPT attacker $A$,

$$\Pr[x \neq x' \ \wedge \ f_{0,PK}(x) = f_{1,PK}(x') \mid PK \leftarrow \mathsf{Gen}(1^k), (x,x') \leftarrow A(PK, 1^k)] \leq \mathsf{negl}(k)$$

♢

Notice, we will omit $PK$ from our notation, to avoid messiness, and also assume $|\mathcal{M}| \approx 2^n$ for concreteness. Before giving examples of CFPs, let us right away construct a CRHF from a family of CFPs $\{(f_0, f_1)\}$. Given any (fixed, but possibly huge) parameter $L$, view the message $M$ as a pair $(x,m)$, where $x \in \mathcal{M}$ and $m = m_1 \ldots m_L$, and let

$$h(x,m) = f_{m_L}(\ldots f_{m_2}(f_{m_1}(x))\ldots)$$

We claim that $h$ is a CRHF from roungly $L+n$ bits to $n$ bits. In particular, it already compresses for $L = 1$. Also notice we can process the input in an on-line manner. Now, take any two messages $(x,m) \neq (x',m')$ and assume $h(x,m) = h(x',m') = z$. We have two cases:

**Case 1:** Assume $m = m'$. Notice, although $f_0^{-1}$ and $f_1^{-1}$ might not be efficient, they are well defined mathmatically. In particular, $x = f_{m_1}^{-1}(\ldots f_{m_L}^{-1}(z)\ldots)$, $x' = f_{m'_1}^{-1}(\ldots f_{m'_L}^{-1}(z)\ldots)$. Thus, if $m = m'$, we get $x = x'$ as well.

**Case 2:** Assume $m \neq m'$. Let $i$ be the smallest index such that $m_i \neq m'_i$ (but $m_j = m'_j$ for $j > i$). Say, $m_i = 0$, $m'_i = 1$. Let $z' = f_{m_{i+1}}^{-1}(\ldots f_{m_L}^{-1}(z)\ldots)$, $x_0 = f_{m_{i-1}}(\ldots f_{m_1}(x)\ldots)$,

$x_1 = f_{m'_{i-1}}(\ldots f_{m'_1}(x)\ldots)$. Then, $f_0(x_0) = z' = f_1(x_1)$, so $(x_0, x_1)$ form a claw (which is efficiently computable, see above formula).

Notice, the construction easily generalizes for any suffix-free messages. Thus, by appending a block containing $\langle L \rangle$, we can maintain suffix-freeness, and still preserve the on-line nature of the function!

Examples of CFPs. We now argue that standard one-way permutation constructions, such as exponentiation mod $p$, squaring mod $n$, and RSA, easily yield CFPs. We demondtrate it for exponentiation, leaving the other two cases as simple exercises (done analogously to exponentiation).

The public key $PK$ here consists of a prime $p$, a generator $g$ of any subgroup $\mathcal{G}$ of $\mathbb{Z}_p^*$ (including $\mathbb{Z}_p^*$ itself) where discrete log is hard, and a random element $y \in \mathcal{G}$. We define $f_0(x_0) = g^{x_0} \bmod p$, and $f_1(x_1) = y \cdot g^{x_1} \bmod p$. Now, if $f_0(x_0) = f_1(x_1)$, then $g^{x_0} = yg^{x_1} \bmod p$, meaning that $y = g^{x_0 - x_1} \bmod p$, meaning that $(x_0 - x_1)$ is a discrete log of $y$. Since $y$ was random, it should be hard to compute discrete log of $y$. Thus, it must be infeasible by the attacker to compute $x_0, x_1$ above.

Notice, this construction is elegant, but very inefficient: it roughly requires an exponentiation per bit of the input. Also, for $\mathcal{G} = \mathbb{Z}_p^*$, we indeed have compression starting from $L = 1$. However, for smaller subgroups $\mathcal{G}$, the output is an element of $\mathcal{G}$, and it might not be easy to compress it to $\log |\mathcal{G}|$. Thus, we prefer to either use $\mathbb{Z}_p^*$ itself, or let $p = 2q + 1$, and consider the prime order $q$ subgroup of quadratic residues mod $p$ (like we did for ElGamal). This is compressing for any $L \geq 2$, and can be made compressing even for any $L = 1$ using the bijection between $QR(p)$ and $\mathbb{Z}_q$ we introduced in the lecture covering the ElGamal encryption.

### 4.3 Optimized Constructions using Number Theory

We can optimize the above generic construction using the specific algebraic properties of concrete one-way permutations, such as exponentiation and RSA. We only do it for the former, mentioning that the other examples can also be optimized.

Construction from Discrete Log. For that, let us recall the CFP construction based on the discrete log, and let us look at it when $L = 1$. In this case, the input to $h$ is a message consisting of a bit $b$ and an integer $x \in \{1 \ldots |\mathcal{G}|\}$, and the output is $h(b, x) = y^b g^x \bmod p$ (yielding $g^x$ for $b = 0$ and $yg^x$ for $b = 1$). For elegancy sake, we will only consider the cases $\mathcal{G} = \mathbb{Z}_p^*$, in which case the output lies in $\mathbb{Z}_p^* \equiv \mathbb{Z}_{p-1}$, and $\mathcal{G} = QR(p)$, where $p = 2q + 1$ and $q$ is prime, in which case we can compress the output to lies in $QR(p) \equiv \mathbb{Z}_q$.

Looking at the formula above, one may wonder why we restrict $b$ to be a bit. Indeed, let us not assume that $b$ is a bit, and see if we can argue if $y^b g^x \bmod p$ is a CRHF. Take any $(b, x) \neq (b', x')$ such that $y^b g^x = y^{b'} g^{x'}$. This means $y^{b-b'} = g^{x'-x}$. Now, if $b = b'$, then $x = x'$, so we conclude that $b \neq b'$. We would then like to conclude that the discrete log of $y$ is equal to $(x' - x) \cdot (b - b')^{-1} \bmod |\mathcal{G}|$. But this is only true if $b - b'$ is relatively prime to $|\mathcal{G}|$. For $\mathcal{G} = \mathbb{Z}_p^*$, $|\mathcal{G}| = p - 1$, which is composite, so we cannot elegantly conclude that $(b - b')$ is invertible. There are ways to deal with this problem, but they are messy. Instead, we will look at the simpler case of $\mathcal{G} = QR(p)$, where $p = 2q + 1$. In this case, $|\mathcal{G}| = q$ is prime, so we can indeed compute the discrete log of $y$ as $(x' - x) \cdot (b - b')^{-1} \bmod q$.

In particular, we can make both $x, b \in \mathbb{Z}_q$. However, the result will be more "symmetric" if we rename $x, b$ into $x_1, x_2$ and $g, y$ into $g_1, g_2$. We get the following elegant family of two-to-one CRHF. Choose random $k$-bit prime $p = 2q + 1$, where $q$ is prime. Let $\mathcal{G} = QR(p)$, and notice that $\mathcal{G} \equiv \mathbb{Z}_q$ (with efficient mapping both ways, see ElGamal lecture). Under this isomporphism, define the following hash function $h_{p,g_1,g_2} : \mathbb{Z}_q^2 \to \mathbb{Z}_q$. The public key of $h$ consists of $p$ and two random generators $g_1, g_2 \in \mathcal{G}$. Then, for $x_1, x_2 \in \mathbb{Z}_q$, let $h_{p,g_1,g_2}(x_1, x_2) = g_1^{x_1} g_2^{x_2} \bmod p$ (with the result in $\mathcal{G}$ mapped back to $\mathbb{Z}_q$).

**Theorem 4** *$\mathcal{H} = \{h_{p,g_1,g_2}\}$ above is a two-to-one CRHF from roughly $2k$ to $k$ bits, under the discrete log assumption.*

Notice, one can naturally extend the compression ratio to $t$-to-1 of this construction, by using more generators $g_1 \dots g_t$ (homework?). Also, one can have similar optimized constrctions for the RSA and squaring functions.

Direct construction from factoring. Of course, there are other constructions of CRHFs. Here we mention one simple one, based on factoring. Here one chooses the modulus $n = (2p + 1)(2q + 1)$. Also, as part of the public key, one chooses a random $y \in QR(n)$. Notice, $|\mathbb{Z}_n^*| = 2p \cdot 2q = 4pq$, and $|QR(n)| = |\mathbb{Z}_n^*|/4 = pq$. Thus, the order of $y$ is $pq$ (with high probability). Now, we define the following functions $h$ over the integers $m \in \mathbb{Z}$: $h(m) = y^m \bmod n$. Now, if $h(a) = h(b)$, and $a \neq b$, then $a - b$ must divide $pq$, which is the order of $y$, and be different from 0 (since $a \neq b$). Hence, $4(a - b)$ must divide $\varphi(n)$. However, it is well known that a non-zero multiple of $\varphi(n)$ is enough to factor $n$.

# 5 Digital Signatures

The remainder of this lecture is dedicated to *public-key signature schemes (*PKS*)*, which are the public-key counterparts of the *message authentication codes (*MAC*)* that we studied earlier.

Motivation and intuition. The use of signatures as a form of authentication in the "real world" is very old and widespread. It is based on the assumption that it is very hard to emulate one's handwriting well enough or to modify a document so that differences cannot be detected. If one accepts that, signing is then a very efficient way of ensuring that a given public document bears one's approval.

"Physical" signatures must be reproducible by the signer (that is, every person must have a definite procedure for signing as often as needed) and recognizable by others (also by means of some definite procedure) . Their usefulness comes from the fact that lots of entities (e.g. the government, banks, credit card companies, family and friends) can recognize what one's signature looks like (i.e. one's signature is known by the *public*) and yet they cannot forge it.

In this class we discuss the computational counterpart of "physical" signatures, which are called *digital signatures*. Those are intended to provide the *sender* with the means to authenticate his/her *messages* (here understood to be any information he/she intends to make available to the world) in a way that *can be checked by anyone* but that *cannot be copied by others*.

The message authentication codes (MAC) that we studied last class do not quite fit into the niche of digital signatures. For they are *secret-key authentication schemes* and implicit in that concept is that all parties that share the secret information (which is necessary for authenticity verification) must be trustworthy if one does not want to lose all hope for security. It is then clear that anything that deserves the name *digital signature* should be a *public-key authentication scheme*: even people in which one does not trust completely should be able to check the authenticity of one's signature. That is, one does not want to impose any restrictions on the parties that may want to verify the authenticity of one's signature. Of course, the signing algorithm must use secret information (that is, a *secret key*), which roughly corresponds to one's unique way of signing.

## 5.1 Basic Definition

In this subsection we define the notion of a *public-key signature scheme* as a public-key analog of MAC and then present a definition of security for it. $\mathcal{M}$ is the message space (e.g. $\mathcal{M} = \{0,1\}^k$ or $\mathcal{M} = \{0,1\}^*$).

DEFINITION 3 [Public-Key Signature Scheme] A **Public-Key Signature Scheme (PKS)** is a triple (Gen,Sign,Ver) of PPT algorithms:

a) The key generating algorithm Gen outputs the secret (private) and verification (public) keys: $(SK, VK) \leftarrow \mathsf{Gen}(1^k)$.

b) The message signing algorithm Sign is used to produce a signature for a given message: $\sigma \leftarrow \mathsf{Sign}_{SK}(m)$, for any $m \in \mathcal{M}$.

c) The signature verification algorithm checks the correctness of the signature: $\mathsf{Ver}_{VK}(m, \sigma) \in \{accept, reject\}$

The correctness property must hold: $\forall m, \quad \mathsf{Ver}_{VK}(\mathsf{Sign}_{SK}(m)) = \text{accept}.$ $\diamondsuit$

**Remark 1** *One can also consider* **stateful** PKS*; we shall encounter those towards the end of the lecture.*

**Remark 2** *We shall adopt the convention that $VK$ is a substring appended at the end of $SK$ (i.e. $SK$ contains in $VK$) whenever this is useful. Of course, this entails no loss of generality.*

As in the case of a MAC, we can consider the notions of existential or universal unforgeability against $VK$-only, random-message or chosen-message attacks. To assume that an adversary might be able to query the receiver of the messages to check the validity of given pairs $(m, \sigma)$ makes no difference in the present case, as long as the adversary has the public key, he can test that by himself. Therefore, the natural counterpart of the standard notion of security for MAC in the present context is:

DEFINITION 4 [Standard notion of security for PKS] A PKS $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is said to be secure, that is, existentially unforgeable against chosen-message attack (CMA) if for all PPT $A$ we have that

$$\Pr(\mathsf{Ver}(m, \sigma) = accept \mid (SK, VK) \leftarrow \mathsf{Gen}(1^k), (m, \sigma) \leftarrow A^{\mathsf{Sign}_{SK}}(VK)) \leq \mathsf{negl}(k)$$

where $A$ cannot query the oracle $\mathsf{Sign}_{SK}$ on the message string $m$ it outputs. $\diamondsuit$

WEAKER NOTIONS. We will also study weaker notions of signatures as we try to satisfy the ambitious definition above. They vary according to the attack capabilities and the goal of the attacker:

- **Capabilities of $\mathcal{A}$.** Currently $\mathcal{A}$ is capable of launching chosen message attack: i.e., he has unrestricted access to the signing oracle. We can place a restriction of the number of times $q$ that $\mathcal{A}$ is allowed to call the signining oracle. Currently, $q = \mathsf{poly}(k)$. If $q = 0$, we get *no message* or $VK$*-only* attack. When $q = 1$ (this is an important case we study later), we get what is called *one-time signature*: the signer can securely sign at least one message and be sure no other signature can be forged. One can also restrict adaptivity of $\mathcal{A}$ (i.e., $\mathcal{A}$ cannot select the messages whose signature he sees, or has to select all messages at once), but we will not study these variants.

- **Goal of $\mathcal{A}$.** Currently, the goal of $\mathcal{A}$ is *existential unforgeability*. Namely, $\mathcal{A}$ succeeds as long as he forges a new signature, irrespective of hoe "ridiculous" the message $m$ he is forging is. A more ambitious goal is to forge a signature of *any given message* with non-negligible probability. There are two flavors of it. More formally, $\mathcal{A}$ is given a *random* message and succeeds if he can sign this message with non-negligible probability. A signature scheme secure against this variant is called *universally unforgeable*. Clearly, existential unforgeability is much more desirable than universal unforgeability.

## 5.2 "Hash-then-Sign" Method

We develop the following simple "hash-and-sign" method, which illustrates that, using CRHFs, we only need to construct secure signature schemes on "short" domains, and automatically get secure signatures on "long" domains. The lemma below work for both regular ("many-time") and one-time signature scheme: the former case being extensively used in practice, and the latter will be used by us in the the Naor-Yung construction from the next lecture. For concreteness, the rpoofs below is for regular (multi-time) signature scheme, since the one-time case will be a special case.

**Lemma 3 (Secure schemes with CRHF)** *If $\mathcal{H} = \{h_s\}$ is a* CRHF *and* $\mathsf{SIG}' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Ver}')$ *is a (one-time) secure signature scheme for $\ell$-bit messages, then the signature scheme* $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *defined below is (one-time) secure for $L$-bit messages:*

*a)* $SK = SK'$, $VK = (VK', h)$, *where* $(SK', VK') \leftarrow \mathsf{Gen}'(1^k)$ *and* $h \leftarrow \mathcal{H}$.

*b)* $\mathsf{Sign}_{SK}(m) = \mathsf{Sign}'_{SK'}(h(m))$.

*c)* $\mathsf{Ver}_{VK}(m, \sigma) = \mathsf{Ver}'_{VK'}(h(m), \sigma)$.

**Proof:** Assume that Lemma 3 is false for some $\mathsf{SIG}'$, that is, there exists an adversary $A$ such that

$$\Pr(\mathsf{Ver}'_{SK'}(h(m), \sigma) = 1 \mid (SK', VK') \leftarrow G(1^k), h \leftarrow \mathcal{H}, (m, \sigma) \leftarrow A^{\mathsf{Sign}_{SK'}}(VK', h)) = \varepsilon$$

where $A$ queried the oracle on messages $m_1 \dots m_t$, and, when successful, outputs the signature $\sigma$ of $m \notin \{m_1, \dots, m_t\}$. Then at least one of the following events happens with non-negligible probability $\varepsilon/2$:

(a) $h(m) \in \{h(m_1), \dots, h(m_t)\}$.

(b) $h(m) \notin \{h(m_1), \dots, h(m_t)\}$.

In case (a), we can break the collision-resistance property of $\mathcal{H}$. Indeed, it implies that for some $i$, $h(m) = h(m_i)$, while by assumption $m \neq m_i$, so $m$ and $m_i$ form a collision. In case (b), we break the security of our original signature $\mathsf{SIG}'$. Indeed, $h(m)$ is a "new message" w.r.t. $\mathsf{SIG}'$, since $A$ only saw $\mathsf{Sign}(m_i) = \mathsf{Sign}'(h(m_i))$, so $A$ managed to forge a new signature. Translating this into a formal proof (i.e., building the actual $B$ breaking $\mathsf{SIG}'$) is straightforward. Indeed, $B$ picks its own $h \leftarrow \mathcal{H}$, and simulates oracle calls to $\mathsf{Sign}(m_i)$ by oracle calls to $\mathsf{Sign}'(h(m_i))$. When $A$ forges $(m, \sigma)$, $B$ outputs its own forgery $(h(m), \sigma)$. □

# 6 Advanced: Universal One-Way Hash Functions

**This material is optional and was not covered in class. You can skip it and move to the next lecture.**

DEFINITION 5 A family of functions $\mathcal{H} = \{h_{PK} : \mathcal{M}(PK) \to \mathcal{R}(PK)\}$ generated by $\mathsf{Gen}$ is called a family of *universal one-way hash functions* ($\mathsf{UOWHF}$s) if (a) $|\mathcal{M}| > |\mathcal{R}|$, (b) $h_{PK}$ is efficiently computable for any $PK$, and (c) for any $\mathsf{PPT}$ attacker $A = (A_1, A_2)$,

$$\Pr[x \neq x' \wedge h_{PK}(x) = h_{PK}(x') \mid (x, st) \leftarrow A_1(1^k), PK \leftarrow \mathsf{Gen}(1^k), x' \leftarrow A_2(PK, st)] \leq \mathsf{negl}(k)$$

♢

## 6.1 "Hash-then-Sign" with UOWHFs

**Lemma 4 (Secure schemes with UOWHF)** *If $\mathcal{H} = \{h_s\}$ is a $\mathsf{UOWHF}$ and $\mathsf{SIG}' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Ver}')$ is a (one-time) secure signature scheme for $(\ell + p)$-bit messages, then the signature scheme $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ defined below is (one-time) secure for $L$-bit messages:*

*a) $SK = SK'$, $VK = VK'$, where $(SK', VK') \leftarrow \mathsf{Gen}'(1^k)$.*

*b) $\mathsf{Sign}_{SK}(m) = (h, \mathsf{Sign}'_{SK'}(h \circ h(m))$, where $h \leftarrow \mathcal{H}$ and $\circ$ is concatenation.*

*c) $\mathsf{Ver}_{VK}(m, (h, \sigma)) = \mathsf{Ver}'_{VK'}(h \circ h(m), \sigma)$.*

**Proof:** Assume that Lemma 4 is false for some $\mathsf{SIG}'$, that is, there exists an adversary $A$ such that

$$\Pr(\mathsf{Ver}'_{SK'}(h \circ h(m), \sigma) = 1 \mid (SK', VK') \leftarrow G(1^k), (m, (h, \sigma)) \leftarrow A^{\mathsf{Sign}_{SK'}}(VK')) = \varepsilon$$

where $A$ queried the oracle on messages $m_1 \dots m_t$, and, when successful, outputs the signature $(h, \sigma)$ of $m \notin \{m_1, \dots, m_t\}$. Let also $(h_i, \sigma_i)$ denotes the signature of $m_i$ returned by

the oracle (i.e., each $h_i$ is truly random, even though $h$ used in the forgery could be chosen by $A$ arbitrarily). Then at least one of the following events happens with non-negligible probability $\varepsilon/2$:

(a) $h \circ h(m) \in \{h_1 \circ h_1(m_1), \ldots, h_t \circ h_t(m_t)\}$.

(b) $h \circ h(m) \notin \{h_1 \circ h_1(m_1), \ldots, h_t \circ h_t(m_t)\}$.

In case (a), we can break the universal one-wayness property of $\mathcal{H}$. Indeed, it implies that for some $i$, $h = h_i$ and thus $h_i(m) = h_i(m_i)$, while by assumption $m \neq m_i$. Thus, if we set $x_0 = m_i$, we get that $h = h_i$ was chosen at random *after* $x_0$ was chosen by $A$, and we can set $x_1 = m$, thus creating a collision $x_0 \neq x_1$ for this randomly selected $h_i$. Translating this into a formal proof is easy and is omitted.

In case (b), we break the security of our original signature $\mathsf{SIG}'$. Indeed, $h \circ h(m)$ is a "new message" w.r.t. $\mathsf{SIG}'$, since $A$ only saw $\mathsf{Sign}'(h_i \circ h_i(m_i))$, so $A$ managed to forge a new signature. Translating this into a formal proof (i.e., building the actual $B$ breaking $\mathsf{SIG}'$) is straightforward. Indeed, $B$ simulates oracle calls to $\mathsf{Sign}(m_i)$ by picking a random $h_i \leftarrow \mathcal{H}$ and, getting $\sigma_i = \mathsf{Sign}'(h_i \circ h_i(m_i))$ from its own oracle, and returning $(h_i, \sigma_i)$. When $A$ forges $(m, (h, \sigma))$, $B$ outputs its own forgery $(h \circ h(m), \sigma)$. □

## 6.2 Composition for UOWHF's

The situation is a bit more difficult with UOWHF's. The reason is the following. Assume $\mathcal{H}'$ (build from $\mathcal{H}$ using some tree, where we are not specifying yet how to select the $h_i$'s) is not a UOWHF. Then some $A$ can specify $x \in \{0,1\}^L$, and after $h' \in \mathcal{H}'$ is selected, $A$ can collisde $x$ with some $x'$. We would like to construct $B$ that breaks the fact that $\mathcal{H}$ is UOWHF. First, $B$ must select some $x_i \in \{0,1\}^\ell$ *before* $h = h_i$ is selected. Probably, $B$ should use $A$ to get $x \in \{0,1\}^L$. The question is: how to use $x$ to produce the needed $x_i$? If $B$ randomly selects $h'$, then it can use $A$ to find a local collision $(x_i, x_i')$ to some $h_i \in \mathcal{H}$ (by Lemma 1). But then it's too late: $h = h_i$ is already selected, and $B$ has to compute $x_i$ *before* $h_i$ is selected. Well, $B$ can guess the internal node $i$ that will produce the collision (and has non-negligible chance of being correct). Thus, if $B$ can use $x$ to compute the input $x_i$ to the $h_i$ *without selecting* $h_i$ yet, we will be done. But if we use the same $h$ at all nodes $i$, $B$ cannot compute this $x_i$ since it will have to select $h = h_i$ for that!

On the other extreme, if all the $h_i$ are independently sampled from $\mathcal{H}$, $B$ can succeed with ease. Indeed, it only selects the random $h$'s that are needed to compute the input to $h_i$, without yet selecting $h_i$. This way it can compute $x_i$, then when a random $h$ is selected, it can set $h_i = h$, select the remaining functions, and then use $A$ with Lemma 1 to find the collising $x_i'$ (provided its guess for $i$ is correct). The argument above is correct, but the reduction is very wasteful in terms of the key: for each internal node $i$ we have to choose a new $h_i$! Can we do better? The answer is positive. Notice, in the argument above we only ran into trouble if the same $h$ was used between a node $i$ and some of its descendant $j$: $h_i = h_j$. In this case, if $i$ was the local node "resposible" for the "global" collision, computing the input $x_i$ to node $i$ (from the global input $x$) requires to compute $h_j = h_i$ along the way. And $B$ cannot do this since $h_i$ cannot be selected as of yet. On the other hand, if no node shares the same seed with its descendant, we can easily complete the argument, as before.

This suggests the following more efficient way to select $h$'s. Assume our tree has depth $d$. Then we must[1] use at least $d$ distinct independent $h$'s. On the other hand, $d$ such $h$'s indeed suffice: simply use a different $h$ at each level of the tree, and let all the nodes at depth $i$ use the same $h_i$. We get

**Theorem 5** *Let $\mathcal{H}$ be a* UOWHF *family of fucntions from $\ell_1$ to $\ell$ bits and key length $p$. Assume $\mathcal{H}'$ from $L$ to $\ell$ bits is constructed from $\mathcal{H}$ by using a "legal" tree $T$ of depth $d$ by selecting an independent function $h_i$ for each level of $T$. Then $\mathcal{H}'$ is a* UOWHF *family with key length $p' = dp$, where evaluation of $h'$ takes $(L-\ell)/(\ell_1-\ell)$ evaluations of various $h_i \in \mathcal{H}$. For concrete examples,*

- *If $\ell_1 = \ell + c$ for some $c > 0$ and $T$ is a path of depth $n$, we can achieve $L = nc + \ell$ and $p' = pn = \frac{p(L-\ell)}{c}$, where evaluation of $h' \in \mathcal{H}'$ takes $n = \frac{L-\ell}{c}$ evaluations of $h \in \mathcal{H}$. In paricular, if $c = 1$ we can get $L = 2\ell$, $p' = \ell p$ using $\ell$ evaluations of $h$.*
- *If $\ell_1 = 2\ell$ and $T$ is a CBT, we can achieve $L = 2^d\ell$ and $p' = d\ell$, where evaluation of $h' \in \mathcal{H}'$ takes $(2^d - 1) = \frac{L}{\ell} - 1$ evaluations of $h \in \mathcal{H}$.*
- *Combining the above, if $\ell_1 = \ell + c$, we can achieve $L = 2^d\ell$, and $p' = \frac{p\ell d}{c} = \frac{p\ell \log(L/\ell)}{c}$ where evaluation of $h' \in \mathcal{H}'$ takes $\frac{\ell(2^d-1)}{c} = \frac{L-\ell}{c}$ evaluations of $h \in \mathcal{H}$.*

The above result says that if two trees $T_1$ and $T_2$ yield the same output length $L$, we should select (provided we use our technique) the tree with smaller depth. More precisely, given $\ell$, $\ell_1$ and $L$, we should select the smallest depth "legal" tree with this parameters. It is easy to see that $d = \Omega(\log L/\ell)$ (proof omitted), so the last item in Theorem 5 is nearly optimal. In fact, we mainly care about the dependence of our key size on the input length $L$ (in applications, $\ell_1$, $\ell$ and $p$ can be thought as fixed). Thus, the optimal dependance of of $d$ on $L$ is logarithmic. We summarize this in

**Corollary 5** *Given a fixed* UOWHF *$\mathcal{H}$ with fixed parameters $\ell_1, \ell, p$, we can build a* UOWHF *$\mathcal{H}'$ with input size $L$ and output size $\ell$, so that the key size $p'$ of $\mathcal{H}'$ grows proportionally to* $\log L$.

We remark that the above composition of UOWHF's is not the best that one can do, but it is nearly optimal, and certainly good enough for our purposes.

### 6.3 Construction of UOWHF

It turns out that UOWHF's are equivalent to OWF's. The fact that they imply OWF's is left as a homework. The converse implication from OWF's is more interesting, but also much more difficult. We will give a simpler construction from OWP's instead. As we stated, the construction will shrink by only 1 bit: $L = k$, $\ell = k - 1$. Even that will be non-trivial, but we will later see how to have a better tradeoff.

So let $f$ be a fixed OWP (or, more generally, it can be chosen at random from a family of OWP's and fixed). We will use the following auxiliary function family $Chop = \{g_a :$

[1] Meaning if we want to follow this specific proof technique. Of course, there could be, and in fact *there are*, other composition techniques for UOWHF's.

$\{0,1\}^l \to \{0,1\}^{k-1}\}$. This family will be defined over the finite field $F$ having $2^k$ elements (if this is too abstract, you can think for of $F$ as being $\mathbb{Z}_p$, where $p$ is some $k$-bit prime). Here each function $g_{a,b}$ will be given by a non-zero element $a \in F\backslash\{0\}$, and will be defined as $g_a(y) = chop(ay)$, where $ay$ is computed in $F$ (and takes $k$ bits to represent), while $chop(ay)$ simply deletes the last bit of the representation of $ay$, thus truncating it to the needed $(k-1)$ bits. This might seem complicated, but we will only be using the following two elementary properties of the family *Chop*.

(1) For every $z \in \{0,1\}^{k-1}$, and every $g_a \in Chop$, there are exactly two distinct points $y_0$ and $y_1$ such that $g_a(y_0) = g_a(y_1) = z$.

(2) The following, very strange method, nevertheless select a *uniformly random* function $g_a \in Chop$. First, fix *any* adversarilly chosen value $y_0 \in \{0,1\}^k$. Then select $y_1$ *at random*. Next, choose $a$ at random, but *subject to* $g_a(y_0) = g_a(y_1)$. More precisely, if $y_0 = y_1$, choose random non-zero $a$, else set $a = 1/(y_0 - y_1)$. Output $g_a$ as your function.

To verify (1), notice that $y_0 = z0/a$ and $y_1 = z1/a$, where $z0$ and $z1$ are completions of $z$ corresponding to the chopped bit being 0 or 1. Thus, all the functions $g_a$ are 2-to-1. To verify (2), first notice that $g_a$ is uniform provided the unlikely event $y_0 = y_1$ happens (in any event, we can ignore this event since it happens with negligible). Else, if $y_0 \neq y_1$, we get $a = 1/(y_0 - y_1)$ is also random and non-zero since $(y_0 - y_1)$ is random and non-zero. To see that the latter indeed has $g_a(y_0) = g_a(y_1)$, notice that $chop(ay_0) - chop(ay_1) = chop(ay_0 - ay_1) = chop(1) = 0$.

Now we can construct our UOWHF $\mathcal{H} = \{h_a : \{0,1\}^k \to \{0,1\}^{k-1}\}$, where $h_a(x) = g_a(f(x))$ (recall that $f$ is a OWP).

**Theorem 6** *$\mathcal{H}$ above is a* UOWHF.

**Proof:** Assume $\mathcal{H}$ is not a UOWHF. Thus, there exists $x_0 \in \{0,1\}^k$ and an adversary $A$, such that when $a \neq 0$ is selected at random, $A(x_0, a)$ outputs $x_1 \neq x_0$ such that $g_a(f(x_0)) = g_a(f(x_1))$ with probability $\varepsilon$.

Using this $A$, we construct $B$ that inverts our OWP $f$. $B$ gets an input $y = f(x)$ for a randomly chosen (unknown) $x$. $B$ sets $y_0 = f(x_0)$ and $y_1 = y$. Then $B$ uses property (2) above to sample the function $g_a$ subject to $g_a(y_0) = g_a(y_1) = z$. Notice, since $x$ was random and $f$ is a permutation, then $y_1 = y = f(x)$ is random as well, and hence by property (2) we get that $a$ is random, as is expected by $A$. Now $B$ runs $x_1 \leftarrow A(x_0, a)$. By assumption, with probability $\varepsilon$ we indeed have $g_a(f(x_1)) = g_a(f(x_0)) = g_a(y_0) = g_a(y_1) = z$. Since $x_0 \neq x_1$ and $f$ is a permutation, we have that $y_0 = f(x_0) \neq f(x_1)$. But by property (1), $g_a$ has only two preimages of $z$: namely, $y_0$ and $y_1$. Since $f(x_1)$ is also a preimage of $z$ and is different from $y_0$, it must be equal to $y_1$. Thus, $f(x_1) = y_1 = y$, so $x_1 = x$ indeed. □

## 6.4 Comparing CRHF's and UOWHF's

From what we have seen, we can compare the pros and cons of using CRHF's vs. UOWHF's.

1. UOWHF's are equivalent to OWF's, while CRHF's probably form a stronger assumption of their own. Thus, basing something of UOWHF's is more general.

2. CRHF's give a more efficient hash-then-sign method, since the hash function does not have to be signed.

3. CRHF's are easier to compose than UOWHF's. Namely, the same $h$ can be used all the time. While UOWHF's require the key to grow (only logarithmically though) with the length of the message hashed. Combined with the previous point, this gives even more preference to hash-then-sign using CRHF's.

4. Most number theoretic assumption that we use for constructing OWF's (like factoring or discrete log) actually suffice for provably secure CRHF's as well.

5. In practice, people use a single hash function (like MD5 or SHA-1) and hope "it is collision-resistant". Thus, in practice people anyway assume that what they are using is a "collision-resistant function".

To summarize, in theory the distinction is very important, while in practice assuming CRHF's is OK.

| CSCI-GA.3210-001<br>MATH-GA.2170-001 | **Introduction to Cryptography** | April 18, 2012 |
|---|---|---|

# Lecture 13

*Lecturer: Yevgeniy Dodis* *Spring 2012*

This lecture is dedicated to constructions of digital signature schemes. Assuming the exisitence of CRHFs (or UOWHFs), we know that it is sufficient to construct signature schemes for fixed-length messages (at least as long as the security parameter). However, we will see that such constructions are not easy. We start with an educational construction, based on trapdoor permutations. This construction will be insecure. In fact, it's insecurity will suggest — incorrectly — that it might be impossible to construct a secure signature scheme. Luckily, this conclusion, known as "signature paradox", will turn out to be wrong. We will then construct *one-time signature* schemes, which will allow one to sign at most one message. Then, we give Naor-Yung construction, which shows how to extend a one-time signature into a regular-signature. The construction will crucially use the "hash-then-sign" paradigm, but will be somewhat inefficient. Finally, we will move to more practical secure signature schemes. For that, we will introduce the *random oracle model*, and show how to revive the originally doomed "trapdoor signature" scheme in this model. This scheme is called *full domain hash*, and is extensively used in practice.

## 1 Examples and problems

The purpose of this section is twofold. First, we show how signature schemes that are "somewhat secure" can be designed using trapdoor permutations. Second, by showing that these schemes fail to meet the security standards we set for signatures, we give evidence that designing secure signatures is hard, if possible at all. We also present a convincing but thankfully fallacious "proof" of the non-existence of secure signature schemes.

### 1.1 Examples: trapdoor signature schemes

The subject of this section is a way of building signature schemes from trapdoor permutations.

**RSA Signature.** The idea behind this scheme is the following. We use the *inverse* of the RSA function to produce the signature $\sigma = \mathsf{RSA}^{-1}(m)$ from the message $m$, and those interested in checking it just compute $\mathsf{RSA}(\sigma)$ and compare it with $m$. More precisely (using the notation in the definition of PKS ):

a) $SK = (p, q, d)$ and $VK = (n, e)$ where $p, q$ are random $k$-bit primes, $n = pq$ is the RSA modulus, $e \in \mathbb{Z}^*_{\varphi(n)}$ is the RSA exponent and $d = e^{-1} \bmod \varphi(n)$.

b) $\mathsf{Sign}_{SK}(m) = m^d \bmod n$ (where $m \in \mathbb{Z}^*_n$).

c) $\mathsf{Ver}_{VK}(\sigma) = [\sigma^e = m \bmod n]$ (where $\sigma \in \mathbb{Z}^*_n$).

A $VK$-only attack cannot result in universal forgery with non-negligible probability under the RSA assumption though the following reasoning. Because a successful universal forgery would enable one to find $\mathsf{RSA}^{-1}(m)$ of any (and, thus, of a random) $m$ with probability $\varepsilon$, contrary to the RSA assumption. This implies that under the RSA assumption this scheme is universally unforgeable against $VK$-only attack.

**Rabin Signature.** The idea used in the previous scheme is again used, only substituting the modular squaring function for RSA. That is,

a) $SK = (p, q, g, h)$ and $VK = n$ where $p, q$ are random $k$-bit primes, $g$ generates $\mathbb{Z}_p$, $h$ generates $\mathbb{Z}_q$ and $n = pq$ is the modulus.

b) $\mathsf{Sign}_{SK}(m) = \sqrt{m} \in \mathbb{Z}_n^*$ (where $m \in \mathbb{Z}_n^*$ and any of the four square roots will do).

c) $\mathsf{Ver}_{VK}(\sigma) = [\sigma^2 = m]$ ($\sigma \in \mathbb{Z}_n^*$).

The same argument given for RSA proves that this PKS is universally unforgeable against $VK$-only attacks under the factoring assumption, and therefore even more believable than RSA.

**Signature based on any Trapdoor Function.** It turns out that the above constructions can be generalized for an arbitrary trapdoor function $f$ with trapdoor information $t$ (technically, a *family* of trapdoor functions with an efficient generation algorithm for $(f, t)$)

a) $SK = (f, t)$ and $VK = (f)$.

b) $Sign_{SK}(m) = f^{-1}(m)$, computed using $t$.

c) $Ver_{VK}(\sigma) = [f(\sigma) = m]$.

The previous two examples can be easily put into that framework. We now present a general theorem on the unforgeability of trapdoor signatures.

**Theorem 1 ((In)Security of Trapdoor Signatures against $VK$-only attack)** *If $f$ is a trapdoor family, then the corresponding trapdoor signature scheme is universally unforgeable against $VK$-only attack, but existentially forgeable against $VK$-only attack.*

**Proof:** We prove the first assertion by contradiction. Suppose that $f$ is a trapdoor but the corresponding signature scheme does not have the desired property. That means that there exists some PPT $A$ such that with non-negligible probability $\varepsilon = \varepsilon(k)$

$$\Pr(\mathsf{Sign}_{SK}(m) = \sigma \mid (SK, VK) \leftarrow \mathsf{Gen}(1^k), m \leftarrow \mathcal{M}_k, \sigma \leftarrow A(m)) = \varepsilon$$

Since $\mathsf{Sign}_{SK} = f^{-1}$ and $\mathsf{Ver}_{VK} = f$, we can rewrite this as $\Pr(m = f(\sigma) \mid m \leftarrow \mathcal{M}_k, \sigma \leftarrow A(m)) = \varepsilon$. Moreover, since $f$ is a permutation, if $x \in \mathcal{M}_k$ is random then $m = f(x)$ is random, and therefore $\Pr(f(x) = f(\sigma) | x \leftarrow M_k, \sigma \leftarrow A(f(x))) = \varepsilon$. Therefore, $A$ inverts $f$ with non-negligible probability, which contradicts the fact that $f$ is a trapdoor permutation.

For the second assertion, notice that a PPT adversary $B$ who on input $VK$ outputs $(f(\sigma), \sigma)$ (for some "signature" $\sigma$ he picks) always succeeds in his attack (i.e., $\sigma$ is a signature of "message" $f(\sigma)$). □

Also, it turns out that in some special cases a trapdoor signature may be universally forgeable under the CMA. Consider, for example, trapdoor families for which for all $k$, $\mathcal{M}_k$ is a group and $f : \mathcal{M}_k \to \mathcal{M}_k$ is a group homomorphism. In that case, it is easy for an adversary to find out $\sigma = \mathsf{Sign}_{SK}(m)$ for any $m \in \mathcal{M}_k$ without a direct oracle query for $\mathsf{Sign}_{SK}(m)$. Indeed, for $m = 1$ (i.e. the identity element) we have that $\sigma = 1$, and for $m \neq 1$ it suffices to pick any $m_1 \in \mathcal{M}_k$ that is not 1 or $m$, then compute $m_2 = \frac{m}{m_1}$ (which also different from the identity and from $m$). The adversary can find out $\sigma_1 = \mathsf{Sign}_{SK}(m_1)$ and $\sigma_2 = \mathsf{Sign}_{SK}(m_2)$ by oracle calls and compute $\sigma = \sigma_1\sigma_2$. This shows that such trapdoor families, RSA and Rabin among them, are universally forgeable against chosen-message attacks.

It should be clear by now that the design of "secure" signature schemes is a difficult problem and at this point one might be inclined to believe that it has no solution.

## 1.2 A "Signature Paradox"

Those who subscribe to the pessimism expressed in the final statement of the previous subsection will not have a hard time accepting the following "theorem", which would be a far-reaching generalization of the second statement of the theorem on unforgeability of trapdoor signatures, were it only true.

**Theorem 2 (The fallacious "Signature Paradox")** *If* $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ *is universally unforgeable against* $VK$*-only attack, then it is existentially forgeable against chosen message attacks.*

**Corollary 1 ("No secure signatures")** *There do not exist signature schemes that are existentially unforgeable against* CMA*.*

**Proof:** The corollary follows form the "theorem" because existential unforgeability against CMA implies universal unforgeability against $VK$-only attack. Therefore, a scheme cannot satisfy the former property without also satisfying the latter, and the theorem says that universal unforgeability against $VK$-only attack implies existential forgery against CMA.

As for the "theorem", its "proof" goes as follows. The way one proves that some $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is universally unforgeable under $VK$-only attack is to prove that that property is equivalent to some "hardness" assumption on a problem $X$ that we believe to be true. On one hand, one shows that the "hard" problem $X$ is such that its solution yields an algorithm to break SIG in polynomial time (for instance, if one can factor $n$ then one can take modular square roots mod$n$ and break the Rabin signature scheme with that modulus). On the other hand, one assumes the existence of a PPT $A$ that on input $(VK, m)$ provides a valid signature $\sigma$ for any $m$ and shows (to get a contradiction) that that would imply that there exists a PPT (denoted by $B$) that would use $A$ as a "black box" (i.e. as an oracle) to break the hardness assumption and solve that given instance of $X$ (in the Rabin case, if such an A exists, than one can take square roots, and using that square root algorithm as a black box one can factor the modulus). Thus, the existence of such "universal" $B$ proves that the presumed $A$ does not exist. Notice, however, that $B$ by itself is well-defined: given a good $A$, $B$ would break $X$.

But then, if the adversary can perform CMA, he can use this $B = B(VK)$, only that, in place of using "black box calls" to $A$ to get $\sigma' = A(VK, m)$ (for a given $m$), he uses a CMA instead to get $\sigma = \mathsf{Sign}_{SK}(m)$. Then $B$ can solve $X$ in polynomial time and use that solution to break the signature scheme SIG. Therefore, CMA allows the adversary to substitute attacks to the sender for calls to $A$. For instance, in Rabin's case, that would mean that with a CMA with some well-chosen messages, one would have enough information to factor the modulus $n$ and then break the signature scheme. □

What is wrong with that "proof"? Well, here is one mistake in it. When $B$ uses $A$ as an oracle in the proof by contradiction of the universal unforgeability of SIG, $B$ can make any query he wants to $A$. Among other things, he might make a query of the form $A(VK', m)$, where $VK' \neq VK$. That is, he might try different verification keys, maybe because if one knows $\mathsf{Sign}_{VK'}(m)$ for several different $VK'$ one can make a very good guess of what the solution to $X$ is (who knows?). On the other hand, when we try to turn $B$ into a CMA adversary, we must then commit to a *single* verification key: the one that is randomly chosen by the sender. That is, whenever $B$ launches a CMA on the sender, he (the sender) always uses the $VK$ that he himself has chosen; equivalently (in a more formal language), all oracle calls that $B$ can make to $\mathsf{Sign}_{SK}$ must use the same $SK$ that is in the output of Gen, and it is quite improbable that a $VK' \neq VK$ will work with $SK$ in the right way. Therefore, the last paragraph of that "proof" is wrong.

# 2 One-time secure signature schemes

The falsehood of the "signature paradox" in the last section leaves us with some hope that it might be possible to build secure signature schemes after all. But given all the difficulties we have faced so far, we'd better try to do it one step at a time. Our plan, which we begin to put into practice in this section, is: to build a rather simple scheme that is secure as long as the adversary can only make *one* CMA, and to show how to get a secure scheme out of it.

## 2.1 Lamport's scheme for one bit

One-way functions (OWF) provide a nice way of signing one bit, which we present below.

Definition 1 [Lamport's scheme for 1-bit messages] Using the notation from the definition of PKS, and letting $f : \{0,1\}^* \to \{0,1\}^*$ be a fixed OWF, we define Lamport's scheme for one-bit messages by:

a) $SK = (X_0, X_1)$, where $X_0$ and $X_1$ are drawn randomly and independently from $\{0,1\}^k$, and $VK = (Y_0, Y_1) = (f(X_0), f(X_1))$.

b) $\mathsf{Sign}_{SK}(m) = X_m$ (remember, $m \in \{0,1\}$)

c) $\mathsf{Ver}_{VK}(\sigma, m) = [f(\sigma) = Y_m]$

♢

Lamport's scheme is "one-time secure" in the sense that if the adversary wants to find out what the signature for 1 (say) with only one oracle query, that query must have the

form $\mathsf{Sign}_{SK}(0) = X_0$, which is just a random string and doesn't help him at all in finding out what $\mathsf{Sign}_{SK}(1)$ is. The intuition can be easily transformed into a proof.

**Remark 1** *In fact, Lamport's scheme is "many-time" secure as well for the trivial reason that there are only two messages. So the only non-trivial attack by the adversary is to forge a signature of $b \in \{0,1\}$ given a signature of $(1-b)$, i.e. general security is the same as one-time security.*

We next generalize this to many bits.

## 2.2 Lamport's Scheme for many bits

The generalization for long (say, length $n = p(k)$) messages of Lamport's scheme is:

DEFINITION 2 [Lamport's scheme for $n$-bit messages] Using the notation from the definition of PKS, and letting $f : \{0,1\}^* \to \{0,1\}^*$ be a fixed OWF, we define Lamport's scheme for $\{0,1\}^n$ by:

a) Let $SK = (X_0^1, X_1^1, X_0^2, X_1^2, \ldots, X_0^n, X_1^n)$, where the $X_i^j$'s are drawn randomly and independently from $\{0,1\}^k$, and $VK = (Y_0^1, Y_1^1, Y_0^2, Y_1^2, \ldots, Y_0^n, Y_1^n)$ with $Y_i^j = f(X_i^j)$.

b) $\sigma = \mathsf{Sign}_{SK}(m_1 \ldots m_n) = X_{m_1}, \ldots, X_{m_n}$.

c) $\mathsf{Ver}_{VK}(\sigma_1 \ldots \sigma_n, m_1 \ldots m_n) = [\forall i \in \{1, \ldots, n\}, \quad f(\sigma_i) = Y_{m_i}]$.

$\diamondsuit$

What Lamport's scheme does is it builds two tables, one for signing (in which entry $(i,j)$ corresponds to the block that is used at the $j^{th}$ position of $\sigma$ if $m_j = i$, that is $X_i^j$) and one for verification (in which entry $(i,j)$ corresponds to the block that is used at the $j^{th}$ position of $\sigma$ if $m_j = i$, that is $Y_i^j = f(X_i^j)$). See the illustration below for $n = 5$.

| bit/position | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 0 | $X_0^1$ | $X_0^2$ | $X_0^3$ | $X_0^4$ | $X_0^5$ |
| 1 | $X_1^1$ | $X_1^2$ | $X_1^3$ | $X_1^4$ | $X_1^5$ |

Table for Signing

| bit/position | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 0 | $Y_0^1$ | $Y_0^2$ | $Y_0^3$ | $Y_0^4$ | $Y_0^5$ |
| 1 | $Y_1^1$ | $Y_1^2$ | $Y_1^3$ | $Y_1^4$ | $Y_1^5$ |

Table for Verification

| bit/position | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 0 | $\mathbf{X_0^1}$ | $X_0^2$ | $\mathbf{X_0^3}$ | $\mathbf{X_0^4}$ | $X_0^5$ |
| 1 | $X_1^1$ | $\mathbf{X_1^2}$ | $X_1^3$ | $X_1^4$ | $\mathbf{X_1^5}$ |

The signature of 01001 is shown in bold.

We notice that an adversary can break the scheme with 2 queries, for if it gets $\mathsf{Sign}(0^n) = X_0^1 \dots X_0^n$ and $\mathsf{Sign}(1^n) = X_1^1 \dots X_1^n$, then it knows all $X_i^j$ and can forge a signature for any given message. However, the scheme is one-time secure in the following sense.

DEFINITION 3 [One-time security for PKS] A PKS $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is said to be one-time secure, that is, existentially unforgeable against CMA with one chosen message query only if for all PPT $A$ we have

$$\Pr(Ver(m,\sigma) = accept \mid (SK, VK) \leftarrow \mathsf{Gen}(1^k), (m,\sigma) \leftarrow A^{\mathsf{Sign}_{SK}}(VK)) \leq \mathsf{negl}(k)$$

where *$A$ $A$ can use the oracle for on at most one query $q$* and cannot output forgery $m = q$. ♢

**Theorem 3 (One-time security of Lamport's scheme)**
*Lamport's scheme is one-time secure provided $f$ is a* OWF*.*

**Proof:** By contradiction. Suppose that there exists a PPT adversary $A$ that violates the definition of one-time security for Lamport's scheme with non-negligible probability $\varepsilon$. We can assume without loss of generality that $A$ makes exactly one oracle call, and that the query and the forgery string both the (expected) length $n$.

We consider the following "experiment" $B$. Given input $y$ that $B$ tries to invert, $B$ runs $\mathsf{Gen}(1^k)$ to get $SK$ and $VK$. It then modifies one of the blocks of the verification key: instead of $Y_i^j$ we now have $Y_i'^j = y$ for some random pair $(i,j)$, and all the other blocks stay the same. Let's call this new verification key $VK'$; note that $VK$ and $VK'$ have the same distribution. Also, $B$ knows the secret key $SK'$ corresponding to $VK'$ except for the value $x \in f^{-1}(y)$ that $B$ tries to extract from $A$. The key observation (that is easy to check) is the following: *If $y = f(x)$ where $x \in \{0,1\}^k$ is random, $VK$ and $VK'$ have the same distribution; moreover, $i$ and $j$ are independent of $VK'$.* The new verification table looks like this (for $i = 1$, $j = 3$ and $n = 5$):

| bit/position | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 0 | $Y_0^1$ | $Y_0^2$ | $Y_0^3$ | $Y_0^4$ | $Y_0^5$ |
| 1 | $Y_1^1$ | $Y_1^2$ | $\mathbf{y}$ | $Y_1^4$ | $Y_1^5$ |

Denote by $q = q_1 \dots q_n$ the string whose signature $A(VK')$ asks the signing oracle (simulated by $B$). If it happens that $q_j = i$, then we fail in our attempt to recover $x$. However, since $i$ is random and $VK'$ is independent of $i$, we have that $q_j \neq i$ with probability $1/2$. In this latter case, $B$ can easily "sign" $q$ for $A$ since it does not need $x \in f^{-1}(y)$ for the signature. Thus, with probability at least $\varepsilon/2$ we get that $A$ outputs a valid message/signature pair $(m,\sigma)$, where $m \neq q$, i.e. $m_1 \dots m_n \neq q_1 \dots q_n$. Hence, there must be at least one index $\ell$ such that $m_\ell \neq q_\ell$. Since $j$ is chosen at random at the view of $A$ so far was independent from $j$, we get that $\ell = j$ with probability $1/n$. Thus, with overall non-negligible probability $\varepsilon/2n$ we have $m_j = i$, and thus $\sigma_j \in f^{-1}(y)$. Therefore, with non-negligible probability $B$ can output this $\sigma_j$ and invert the OWF $f$. □

### 2.3 Long Messages

Although the Lamport scheme allows us to sign arbitrarily long messages (say, of length $n$), this comes at the expense of having a public key of length roughly $2nk$, which is much larger than the length of the message. As we will see, not only is this inefficient, but it is also *insufficient* for many practical applications of one-time signatures: for example, those that need to sign verification keys (and there are many of those, stay tuned)! So how do we construct one-time signatures where the length of the verification key is (much) shorter than the length of the message? The answer is to use the *hash-thn-sign paradigm*! Namely, rather than one-time signing a long $m$ (which might not "fit"), we first hash it down to a short string $h(m)$, and one-time sign $h(m)$. What properties are needed from $h$? As we saw from last class, collision-resistance is enough. In fact, as was mentioned in the optional material, even universal one-wayness is enough, but we will assume CRHFs for simplicity. Also, although the hash-then-sign was stated for "many-time" signatures, it is easy to see from the proof that it works for one-time signatures as well. Notice, our CRHFs construction had a fixed-length public key $pk$, and were capable of hashing essentially unbounded-length messages. Thus, using the hash-then-sign with such CRHFs, we get a one-time signature capable of one-time signing essentially unbounded messages, and having a fixed-length verification key $vk' = (vk, pk)$, where $vk$ is the verification key capable of handling messages whose length is the *output* of our hash function. Assuming the latter is proportional to the security parameter $k$, and that the public key $pk$ for our CRHF is smaller than $O(k^2)$, — which is the case for all our constructions, — we get the following corollary by using Lampert's one-time signature:

**Lemma 2** *Given a* CRHF *whose output size is $O(k)$ and public key at most $O(k^2)$, there exists a one-time signature scheme capable of signing arbitrary (polynomial-)length messages, and having a verification key of size $O(k^2)$.*

**Remark 2** *Similar lemma holds for* UOWHF*s as well. The only caveat is that the public key of our* UOWHF*s (which are not already* CRHF*s) was proportional to $\log L$, where $L$ is the length of the hashed message. Specifically, we knew how to make it roughly $O(k^2 \log L)$, although better constructions are possible. Thus, using* UOWHF*s, to sign messages of length $L$, we get a final verification key of size $O(k^2 \log L)$. For $L > k^2 \log k$, this is smaller than the length of the message.*

## 3 From One-time to Full-fledged Security

Let OT-SIG = (Gen, Sign, Ver) denote any one-time secure signature scheme capable if signing "long-enough" messages. Our question now is: is there a general way of building a secure PKS from OT-SIG? It is *not* enough to divide the message into blocks and sign each block separately: Lamport's scheme did that and yet didn't meet the security standards that we set for ourselves.

In what follows, we describe several natural approaches, eventually leading to a positive answer to this question.

### 3.1 First Attempt: Use Independent Keys

The first natural idea is to choose $t$ independent key pairs $\{(SK_i, VK_i)\}_{i=1}^{t}$ for our one-time signature scheme, and use the $i$-th pair of keys to sign ti $i$-th message $m_i$. In particular, after the $i$-the message is signed, the signer will remember to never reuse the $i$-th secret key again. This "works", but has the following disadvantages:

- **Signer must keep state.** Indeed, the signer must remember which keys were used, to ensure he will never reuse the old key. Thus, we have what is called a *stateful* signature scheme.
- **A-Priori Bounded Number of Messages.** Clearly, one ccan only sign $t$ messages, where $t$ is the parameter which needs to be decided upon at the beginning. Ideally, we would like to sign an arbitrary (polynimail) number of messages.
- **Long Verfication Key.** The length of the verification key $(VK_1 \dots VK_t)$ is proportional to the maximum number of signed messages $t$. Thus, if $t$ is large, so is the verification key.
- **Long Signing Key.** Same as above, but for the signing key.

### 3.2 Second Attempt: Merkle Trees

We start with the simplest pronlem: long verification key $VK = (VK_1, ldots, VK_t)$. Instead, ley $h$ is a collision-resistant hash function, and let $vk = h(VK)$ (plus the description of $h$). This works, but, now, the signer must include $VK$ as part of the signature, since otherwise, the verifier will not know which verification key $VK_i$ to use. This makes the signature size proportional to $t$, which is pretty bad.

A natural attempt would be to only provide $VK_i$ to the verifier as part of the signature. This reduces the length of the signature, but creates another problem: how does the verifier, who only knows $vk = h(VK_1, \dots, VK_t)$, check that $VK_i$ is the correct key, and not some bogus key provided by the attacker? At first, it seems like there is nothing we can do, beside providing the entire $VK$. But then we can remember the idea of the *Merkle tree*, which solves precisely the problem that we have!

Indeed, by using a "bottom-up" complete binary tree to iteravely hash $VK_1, \dots, VK_t$ (using the same $h$), we know that we can prove any particular $VK_i$ by opening only $\log t$ values on the path from $VK_i$ to the root: namely, $VK_i$ itself and all the siblings of the nodes on the path up from $VK_i$ to the root $vk$. This is called *Merkle Signature*. As before, the scheme is stateful, allows to sign an a-priori bounded number of messages $t$, has the secret key proportional to $t$, but now has a constant (i.e., $O(k)$) verification key, and the total signature of size $O(k \log t)$.

### 3.3 Third Attempt: Top-Down Approach

We try a different idea here, which will allow us to get rid of the main problem we faced so far — an a-priori bound on the number $t$ of signed messages. Instead of going bottom-up,

like with Merkle signatures, we will ensure a constant size verification key by going "top-down". We start with using a simple path, latter extending it to a complete binarty tree, and even later taking care of other inefficiencies. The basic idea is due to Naor-Yung.

What we do is the following. Suppose one wants to sign the messages $m_0, m_1 \ldots$ (of appropriate length) in that given order, where there is no bound of $t$. The scheme $\mathsf{SIG}$ that we propose proceeds as follows.

a) First, it gets $(SK_0, VK_0) \leftarrow \mathsf{Gen}(1^k)$. $VK_0$ is the (public) verification key of our new scheme $\mathsf{SIG}$.

b) To sign $m_0$, $\mathsf{SIG}$ gets $(SK_1, VK_1) \leftarrow \mathsf{Gen}(1^k)$, then computes $\sigma_1 = \mathsf{Sign}_{SK_0}(m_0, VK_1)$ (namely, it signs a tuple $[m_0, VK_1]$, where , is some special character that doesn't show up in other places so that we can separate $m_0$ from $VK_1$) and outputs $(\sigma_1, VK_1, m_0)$ as the signature of $m_0$ (for notational convenience, we include the message inside the signature). It also remembers $(\sigma_1, VK_1, m_0)$ for future use.

c) To check whether $(\sigma_1, VK_1)$ is a valid signature for $m_0$ under $\mathsf{SIG}$, the receiver checks if $\mathsf{Ver}_{VK_0}([m_0, VK_1], \sigma_1) = \mathit{accept}$ (i.e. whether $\sigma_1$ is a valid signature for $[m_0, VK_1]$ in the $\mathsf{OT\text{-}SIG}$ scheme.

d) Inductively, to sign $m_i$ (for $i \geq 2$), $\mathsf{SIG}$ gets $(SK_{i+1}, VK_{i+1}) \leftarrow \mathsf{Gen}(1^k)$, computes $\sigma_{i+1} = \mathsf{Sign}_{SK_i}(m_i, VK_{i+1})$ and outputs $(\sigma_{i+1}, VK_{i+1}, m_i \ldots, \sigma_1, VK_1, m_0)$ (i.e. the entire history so far!) as the signature of $m_i$.

e) Finally, to check whether $(\sigma_{i+1}, VK_{i+1}, m_i \ldots, \sigma_1, VK_1, m_0)$ is a good signature of $m_i$, one successively checks all the signatures by $\mathsf{Ver}_{VK_j}([\sigma_{j+1}, VK_{j+1}], m_j)$, and accepts only if the entire chain is valid $(0 \leq j \leq i)$, where the last key $VK_0$ is taken from the public file.

On an intuitive level, this scheme is secure because *no key is used more than once*, and therefore the "one-timeness" of $\mathsf{OT\text{-}SIG}$ is enough. Indeed, if an adversary $B$ attacks the sender with successive chosen messages, each answer it will get will correspond to a different secret key and that will circumvent the original limitations of $\mathsf{OT\text{-}SIG}$. As we sketch below, this is indeed correct. However, notice a crucial requirement for our $\mathsf{OT\text{-}SIG}$: it has to sign messages which are longer than its verification key! Indeed, already at the first level one needs to sign the key $VK_1$ plus the first message using $SK_0$. Luckily, due to Lemma 2 (and remark Remark 2 for $\mathsf{UOWHF}$s), this is no problem!

**Theorem 4** *Provided* $\mathsf{OT\text{-}SIG}$ *can sign messages longer that the length of its verification key, the above (stateful and inefficient) construction is existentially unforgeable against chosen message attack (for messages of corresponding length as explained above).*

**Proof:** The proof is simple, but a bit tedious, so we just sketch the idea (the sketch below can be easily transofrmed into a formal proof).

Say some $A$ asks to sign messages $m_0 \ldots m_t$, gets a chain $(\sigma_{t+1}, VK_{t+1}, m_t \ldots, \sigma_1, VK_1, m_0)$ from the oracle, and forges the signature $(\sigma'_{i+1}, VK'_{i+1}, m'_i \ldots, \sigma'_1, VK'_1, m'_0)$ of some $m'_i \notin \{m_1 \ldots m_t\}$. We claim that there exists and index $j \leq \max(i, t)$ such that "along the way", $A$ produced a forgery $\sigma'_j$ of a "new message" $[m'_j, VK_{j+1}]$ under the key $SK_j$, which contradicts one-time security of the $j$-th one-time signture. The proof is a bit boring:

1. If $[m'_0, VK'_1] \neq [m_0, VK_1]$, then $[m'_0, VK'_1]$ is a new message w.r.t. $VK_0$, and $\sigma_1$ is the forgery.

2. Otherwise (equality so far), if $[m'_1, VK'_2] \neq [m_1, VK_2]$, then $[m'_1, VK'_2]$ is a new message w.r.t. $VK_1 = VK'_1$, and $\sigma_2$ is the forgery.

3. Otherwise (equality so far), if $[m'_2, VK'_3] \neq [m_2, VK_3]$, then $[m'_2, VK'_3]$ is a new message w.r.t. $VK_2 = VK'_2$, and $\sigma_3$ is the forgery.

4. And so on. The point is that since we have $m'_i \notin \{m_1 \dots m_t\}$, at some point $j$ we *must* have inequality: at the worst case, if $i > t$, $[m'_{t+1}, VK_{t+2}]$ is a new message w.r.t. $VK_{t+1}$, since no signatures w.r.t. $VK_{t+1}$ were given to $A$ by its oracle.

Of course, how do we find this $j$, and how do we simulate the run of $A$ with this $j$. Well, we pick $j$ at random from $\{0 \dots T\}$ (where $T$ is the upper bound of $A$'s running time). We generate all the keys on our own, except for the $j$-th key, where we use the given verification key $VK$ whose one-time seurity we want to compromise. The formal proof follows quite easily from the above. □

**Remark 3** *Notice, however, that this construction (known as the* **Naor-Yung** *construction) shows that our explanation of the flaws of the "proof of the signature paradox" is not at all artificial. The scheme* SIG *above uses* Sign *(from* OT-SIG*) as a black box, but each time it does so the secret key is different.*

### 3.4 Improvements: removing state and fixing signature size

The signature scheme above has at least two undesirable features. First, it is stateful. Second, the size of the signature is proportional to the number of messages. It turns out, we can remove these negative features.

We start with the second problem. Notice, our verification procedure can be thought as a long path: first signature authenticates the second, the second — the third, and so on until the last $i$-th signature authenticates the actual $i$-th message signed. Thus, we have a grawing path $VK_0 \to VK_1 \to \dots$. It seems much more economical to use a complete binary tree. Namely, the original $VK = VK_\varepsilon$ authneticates *two* new key: $VK_0$ and $VK_1$. $VK_0$ in turn authenticates $VK_{00}$ and $VK_{01}$, while $VK_1$ — $VK_{10}$ and $VK_{11}$. And so on until some level $k$ (say, our security parameter; we only need the level to withstand the birthday attack, as we shall see). More specifically, imagine the following exponential collection of signatures (kept implicitly): $\mathsf{Sign}_{SK_\varepsilon}(VK_0, VK_1), \dots, \mathsf{Sign}_{SK_x}(VK_{x0}, VK_{x1}, \dots$ until we authnticate all $2k$ nodes $VK_x$, where $|x| = k$. Now, to sign every message $m$ we will use a different root-leaf path $x$ down the tree. Assume we decided on $x = x_1 \dots x_k$ based on $m$ (see how later), the fixed size signature of $m$ is

$$
\begin{array}{rcl}
\mathsf{Sign}_{SK_\varepsilon}(VK_{x_1}, VK_{1-x_1}) & , & VK_{x_1}, VK_{1-x_1}, \\
\mathsf{Sign}_{SK_{x_1}}(VK_{x_1x_2}, VK_{x_1(1-x_2)}) & , & VK_{x_1x_2}, VK_{x_1(1-x_2)} \\
\ldots & , & \ldots \\
\mathsf{Sign}_{SK_{x_1\ldots x_{k-1}}}(VK_{x_1\ldots x_{k-1}x_k}, VK_{x_1\ldots x_{k-1}(1-x_k)}) & , & VK_{x_1\ldots x_{k-1}x_k}, VK_{x_1\ldots x_{k-1}(1-x_k)}, \\
\mathsf{Sign}_{SK_x}(m) & &
\end{array}
$$

We need to address a couple of questions to make this work. We sketch the answers.

- Can we fit two keys inside our one-time signature? the answer is yes if we use a slightly more shrinking CRHF (or UOWHF). So this is not a problem.
- How do we remember these exponentially many keys? The answer is to use a PRF! Namely, we pick a random seed $s$ for a PRF, and use $f_s(x)$ to get the randomness needed to produce the key pair $(SK_x, VK_x)$ for node $x$. Since $s$ is kept secret, these indeed look like properly generated independent signing/verification keys for our one-time signature scheme.
- How do we choose the path $x$ based on $m$, so that they are all distinct?[1] There are several ways to achieve this. The simplest is to choose $x$ at random. Since $k$ is large enough to withstand birthday attack, the probability of reusing the path is negligible. Alternatively, we can use our PRF to extract $k$ pseudorandom bits out of $m$, and let them define $x$ (this makes the signature deterministic!). Yet alternatively, if the message space is $\{0,1\}^k$, we can actually use $x = m$ to achieve uniqueness. Finally, for larger message spaces we can use CRHF's or UOWHF's (the latter should be fresh for every use) to hash our message space into $\{0,1\}^k$.

The above semi-formal sketch gives a construction, which: (1) can be based on OWFs and CRHFs (or even UOWHFs[2]); (2) is stateless; (3) can be even made deterministic; (4) has fixed signature size; (5) can sign arbitrarily large messages (if needed, using hash-then-sign method); (6) is existentially unforgeable under the chosen message attack. In particular, if we use the UOWHFs instead of more powerful CRHFs, since it is known that UOWHFs can be built from OWFs, in principle we get the following (very inefficient) result:

**Corollary 3** *Secure signature schemes exist iff* OWF *'s exist.*

### 3.5 Efficient Signatures?

Unfortunately, the above construction is *extremely* inefficient, and can never be used in practice. Thus, the next question is *what do we do in practice?* Half of the answer we already

[1] Remember, we need distinctness to prove that the scheme is unforgeable; indeed, if we use the same $x$ for $m_1$ and $m_2$, the adversary might recover $SK_x$ after seeing signatures of $m_1$ and $m_2$ with $SK_x$, and then reuse the authentication path $x$ to forge a signature of any other message. We also know that distinctness of all the paths suffices to show security.

[2] For the case of UOWHFs, notice that the messages we need to has can be thought as chosen *before* the new hash key is chosen, so using UOWHFs is OK.

know: using the hash-then-sign paradigm, it suffices to design efficient fixed-message-size signature schemes. But how to design the latter ones efficiently? We give two answers to that:

- **Using Specific Number-Theory Assumptions**. For example, the Cramer-Shoup signature scheme is provably secure under s stronger version of the RSA assumption: given composite $n$ and random $y \in \mathbb{Z}_n^*$ it is hard to extract *any* non-trivial root of $y$. Namely, it is hard to come up with $x \in \mathbb{Z}_n^*$ and $e > 1$ so that $x^e = y \bmod n$. The usual RSA assumption fixes $e$ as a (random) input to the problem. We do not have time to present this signature scheme, since its *proof* is somewhat complicated. Interestingly, this is pretty much *the only provable and efficient* number-theoretic signature scheme up to date.
- **Using Random Oracle Model.** This is the topic of the next section. It turns out that in this model there is a huge variety of very simple-to-state signature schemes (and other cryptographic primitives, like encryption, etc.!) which are: (1) extremely efficient; (2) provably secure. What's the catch? This model assumes something which does not exist... Curious? Read the next section.

# 4 Random Oracle Model and Full Domain Hash

Remember the hash-then-sign methodology. Informally, it said that if $\mathsf{SIG}$ is a "secure" signature scheme, and $h$ is a "well-behaved" hash function, then $\mathsf{SIG}'$ inherits the security of $\mathsf{SIG}$, where $\mathsf{Sign}'(m) = \mathsf{Sign}(h(m))$. Thus, a "good" hash function may succeed in *preserving* the security of our original signature scheme. Let's ask a more ambitious question. Can a "really good" hash function *improve* the security of the signature scheme we started from? To be more specific, consider the first signature scheme that comes to mind and that we originally rejected — trapdoor signature. Here $f$ is a $\mathsf{TDP}$ and $\mathsf{Sign}(m) = f^{-1}(m)$ while $\mathsf{Ver}(\sigma) = [f(\sigma) \stackrel{?}{=} m]$. We saw that this scheme is existentially forgeable under key-only attack. However, let us try to apply the hash-then-sign method hoping that good enough $h$ can make the scheme secure. We get $\mathsf{Sign}(m) = f^{-1}(h(m))$ and $\mathsf{Ver}(\sigma) = [f(\sigma) \stackrel{?}{=} h(m)]$. Rephrasing our question above, *what properties of h (if any) would make the trapdoor signature above existentially unforgeable against the chosen message attack?* From a practical point of view, it seems like having some really good function $h$ indeed improves the security of the trapdoor signature: for example, existential forgery no longer seems possible. But can we prove it?

Before answering this question, let's try some of the candidates for $h$ which we already know about. First, assume $h$ is chosen at random from the family of $\mathsf{CRHF}$'s, and made part of the public key. Unfortunately, this does not seem to be sufficient. Intuitively, to prove the security of a construction based on a trapdoor permutation, the permutation (and thus its inverse) has to be always applied to a (pseudo)random input. In our case, we compute $f^{-1}(h(m))$, which is (pseudo)random only if $h(m)$ is (pseudo)random. The definition of $\mathsf{CRHF}$'s says nothing about pseudorandomness of $h(m)$. In fact, one can design very non-random $\mathsf{CRHF}$ families. Thus, $\mathsf{CRHF}$'s do not seem to suffice. The next attempt will be to choose $h$ from a family of $\mathsf{PRF}$'s. This seems to solve the pseudorandomness of

$h(m)$ issue... but not quite. The problem is, should the seed $h$ to our family be public or private? If it is private, the family is indeed pseudorandom, but then we cannot verify the signatures, since we cannot compute the values $h(m)$. On the other hand, if it is made public, the values $h(m)$ no longer seem pseudorandom! More precisely, keeping $h$ public will never allow us to contradict the security of PRF's (which rely on the seed being secret). Thus, PRF's do not quite work as well.

In fact, no conventional function $h$ (even taken from some function family) can work![3] Indeed, $h$ has to be public for verification, and pseudorandom for security. Do there exist public pseudorandom functions? The answer is no. Once something is public (and efficient), it is never pseudorandom: indeed, predicting $h(m)$ is trivial. So it seems like we did not achieve anything. But what if we *assumed that public truly random functions exist*? And then, assuming $h$ is such a function, can we prove that the modified trapdoor signature is secure?

## 4.1 Random Oracle Model

We will demonstrate in a second that the answer to the above question is indeed positive. But first, let us examine our new model more closely. In this model, called the *Random Oracle Model*, one assumes the existence of a public truly random function $h$. This $h$ is called a random oracle.

The model seems contradictory at first. A skeptic might say: "No wonder we can great things in this model. Since a public function is no longer random, we are assuming something which does not exist. From a false statement, anything can be proven." And indeed, many people appall the RO model. However, it is not as meaningless as one might imagine. First, we are not necessarily proving the *existence* of some objects. Usually, we are basing some specific construction on a "ideal function" $h$, and try to argue if this construction is secure. Second, one can "implement" our function in the following way. Imagine a true "random oracle" $O$ sitting in the sky. Whenever we need some value $h(m)$, we give $O$ a query $m$, and $O$ returns $h(m)$. The oracle $O$ is assumed to be completely trusted: (1) all the values returned are random and independent from each other; (2) the same query $m$ will return the same answer $h(m)$, and (3) $O$ does not cooperate with an adversary, so every "new" value $h(m)$ indeed looks random to everyone (including the adversary). In this sense, the model is actually "implementable". Of course, in real life, the role of $O$ be be played by some publically known function $h$. But what the proof of security in the RO model really says is the following: "If you believe that the only way the adversary uses the knowledge of $h$ is by computing $h$ at points of its choice, and if the function $h$ indeed looks pseudorandom to such a restricted adversary, then the adversary indeed cannot break the security of the system." To put it differently, it rules out at least a certain class of "black-box" attacks. Namely, if the adversary wants to break the system, it really has to look at the details of $h$'s implementation and try to exploit them. If it treats $h$ like a "black-box", it cannot be successful.

To summarize the above discussion, RO model is a very strong assumption, in fact, non-existent. However, the proof in the RO model are not meaningless: they at least show

[3] In fact, there are papers shoing it is very unlikely to base the security of the signature above in the "standard model".

that there is something secure about the system designed, and to find a possible flaw, the adversary must utilize the weaknesses of the public function $h$. In practice, functions like SHA do not seem to have to weaknesses which are easy to exploit, at least up to date.[4] Thus, in practice, scheme secure in the RO model, are "currently secure" in the real life as well. Having said that, one should be very careful when using the random oracle. In particular, to justify a scheme in the RO model, this scheme should satisfy be either significantly simpler/efficient than current schemes without random oracle, or there should be no provable schemes without random oracle that achieve the task at hand. In practice, both cases happen: many things (including signatures and encryption) are very easy to do in the RO model, and for certain advanced cryptographic concepts (i.e., "identity based cryptography" among other), only RO-based solutions are currently known. Below we give the simplest example of using the RO model.

### 4.2 Full Domain Hash

We now come back to the modified trapdoor signature scheme — called the full domain hash — and show its security in the RO model. The proof will show the power of the model.

Recall, $f$ is a $\mathsf{TDP}$, $\mathsf{Sign}(m) = f^{-1}(h(m))$ and $\mathsf{Ver}(\sigma) = [f(\sigma) \stackrel{?}{=} h(m)]$. Intuitively, seeing a signature of some message $m$ corresponds to seeing a random $(x, f(x))$ pair (where $x = f^{-1}(h(m))$ is indeed random). Such pair the adversary can generate by itself. On the other hand, forging a signature of a "new" $m$ corresponds to inverting $f$ at a *random* point $h(m)$, which should be hard since $f$ is one-way. We translate this intuition into a formal proof.

**Theorem 5** *Full domain hash is existentially unforgeable against the chosen message attack, provided $f$ is a* $\mathsf{TDP}$ *and $h$ is modeled as a RO.*

**Proof:** Assume some $A$ produces existentially forgery of the full domain hash with probability $\varepsilon$. Say, $A$ asks signatures of $m_1 \dots m_t$, and forges a signature of $m \notin \{m_1 \dots m_t\}$. In order for $A$ to work, $A$ requires oracle access to RO $h$ (we denote this by writing $A^h$). Without loss of generality, we assume that before asking a signature of $m_i$, $A^h$ asked the oracle the value $h(m_i)$. Also, before producing the forgery for $m$, $A^h$ checked $h(m)$. Clearly, $A^h$ might as well do these things without much loss in efficiency.

Using $A^h$, we construct an adversary $B$ which breaks the one-wayness property of $f$ with non-negligible probability $\varepsilon/q$, where $q$ is the maximum number of questions $A$ asks the random oracle.[5] Naturally, $B$ should somehow simulate the run of $A^h$. But here is a "complication", whose "resolution" really shows the power of the RO model: *$B$ does not have any random oracles!* Indeed, $B$ is supposed to invert a $\mathsf{TDP}$ $f$ in the standard model we were using so far. On the other hand, $A^h$ expects access to RO $h$. Where can $B$ provide $A^h$ with $h$? The answer is, *$B$ simulates the RO bu itself.* Namely, whenever $A^h$ wants to get $h(z)$ for some $z$, $A^h$ now "really aks $B$ for it", and $B$ returns $h(z)$ in place of the oracle. One way for $B$ to do it is to indeed return a random string for every new query of $A^h$

[4] Recently, some attacks on SHA were found, but they are more or less very clever brute-force attacks, which are easily solved by making the output slightly longer and/or increasing the number of rounds.

[5] In turns out that a more careful analysis can improve this pessimistic, but sufficient bound. For clarity, we will not present such better analysis here.

(and return consistent strings for old queries; since the number of queries is polynomial, $B$ can remember all the queries so far). As we will see in a second, however, $B$ can be more creative, provided all the answers are indeed random (indeed, $A$ we do not know if $A$ works well with "non-random" oracles). In other words, $B$ can pick a random string in a possibly more complicated way, than by "simply picking a random string". We will see this in a second.

Let's now describe our $B$. On input $y = f(x)$ for unknown $x$, $B$ sets the public key of $\mathsf{SIG}$ to the desription of $f$, picks a random $j \in \{1 \dots q\}$ (recall, $q$ is the number of question $A^h$ asks to RO) and starts running $A^h$ with this verification key. Whenever $A^h$ asks his $i$-th query $h(z_i)$, $B$ does the following. If $z_i$ was already asked, $B$ answers with the same value as before (i.e., answers consistently). If the query $z$ is "new" and $i \neq j$, $B$ picks a random $x_i$, computes $y_i = f(x_i)$, and claims that $h(z_i) = y_i$. Notice, $y_i$ is indeed random, since $f$ is a permutation, and $x_i$ is random. $B$ then remembers $(x_i, y_i, z_i)$, so that in addition to $y_i$, $B$ also knows the "signature" $x_i$ of $z_i$ (indeed, $f(x_i) = y_i = h(z_i)$!). Finally, if the query is new and $i = j$, $B$ returns its own input $y$ and the value $h(z_j)$. Notice, in the case the "signature" of $z_j$ is the value $x = f^{-1}(y)$ that $B$ is trying to find. Jumping ahead, $B$ hopes $A^h$ will forge a signature $x$ of $m = z_j$.

Next, we have to tell how $A^h$ answers the signing queries of $A^h$. Assume $A^h$ wants to sign the message $m_s$. By our assumption, we assumed that $A^h$ would first ask $h(m_s)$, so $m_s = z_i$ for some $i$. But unless $i = j$ (in which case $B$ halts with failure), $B$ already knows the signature $x_i$ of $z_i = m_s$! Thus, unless $i = j$ $B$ can successfully answers the the signing queries. Finally, it's $A^h$'s turn to output the forgery $(m, \sigma)$. We assumed that $A^h$ asked the value $h(m)$ earlier, so $m = z_i$ for some $i$. Notice, if $\sigma$ is valid and $i = j$, $\sigma = f^{-1}(h(z_j)) = f^{-1}(y)) = x$. Thus, $B$ succeeds in inverting $y = f(x)$ provided $A^h$ forged the signature of $z_j$. Since $j$ was chosen at random, and all the random oracle answers were independent of $j$ (since $y$ and all the $y_i$'s were random), the probability that $A^h$ finds a signature of $z_j$ is at least $\varepsilon/q$, which is non-negligible. This completes the proof. □

We remark on the crucial point of the proof: $B$ has control over how to simulate the random oracle, so that it can later answer the signing quesries of $A$. This is exactly the power of the RO model. We assume that the adversary can only access $h$ in a "black-box" way, so it is legal for $B$ to know $A$'s questions, and to prepare "convenient" answers.

**Remark 4** *We also notice that our reduction lost a pretty significant factor $q$ in the security, where $q$ is the number of hash queries made by the attacker. While the loss of this factor $q$ is unavoidable with general* $\mathsf{TDP}$*s, it turns out that a more clever reductions for specific* $\mathsf{TDP}$*s, including RSA and Rabin. In fact, and* $\mathsf{TDP}$ *$f$ "induced" from a family of* $\mathsf{CFP}$*s $(f, g)$ turns out to be sufficient. For such* $\mathsf{TDP}$*s, the security loss goes from $\varepsilon/q$ to roughly $\varepsilon/q_s$, where $q_s$ is the number of signing queries issued by the attacker (as opposed to $q$, which is the number of hash queries, and which could be much higher in practice).*

We conclude the lecture by pointing out that there are many other simple (and practically importnat) signature scheme designed and analyzed in the RO model.

| CSCI-GA.3210-001<br>MATH-GA.2170-001 | **Introduction to Cryptography** | April 25, 2012 |
|---|---|---|

Lecture 14

*Lecturer: Yevgeniy Dodis* *Spring 2012*

This lecture is on Commitment Schemes. Informally, a commitment scheme abstracts the notion of a "locked box": the contents of the box are hidden (without the key), but can be opened in only one way. First, a formal definition of a non-interactive Commitment Scheme is given along with some explanation. Then some examples of Commitment Schemes are given: based on "committing" encryption, PRG's (thus, OWF's), OWP's, CRHF's, discrete log (Pedersen's commitment), random oracle. Compositions of Commitment schemes are considered including bit-by-bit (ala encryption) and "hash-then-commit" (ala signatures) methods. We also briefly talk about a slightly relaxed notion of commitment, which allows us to use UOWHF's in place of CRHF's, and suffices for some of the applications of commitment. Finally, several applications of Commitment Schemes are given along with a brief explanation of zero knowledge proofs.

# 1 Commitment Schemes

## 1.1 Introduction

Commitment schemes arise out of the need for parties to commit to a choice or value and later communicate that value to the other parties involved in such a way that is fair to all the parties. The main problem here is that we do not want one party to find out about any other party's commitment before the latter opens this value itself. On the other hand, we do not want a party to be able to open its commitment in multiple ways (then, there is no point in "committing" in the first place). Therefore, we want our Commitment Scheme to somewhat resemble a locked box that contains some value. This locked box is given to the parties but it does not reveal anything about the commitment contained in it until the key for the locked box is released so the parties can open it. The "digital" implementation of this locked box however introduces the possibility for the locked box to contain multiple values (i.e., have several valid "keys" that open it in different ways). Therefore we have to ensure that each locked box can only hold one value.

We only consider so called "non-interactive" schemes, where all the communication goes from the sender to the receiver.[1] We will omit the word non-interactive from most of the discussion though. A bit more formally, a *Commitment Scheme* transforms a value $m$ into a pair $(c, d)$ here $c$ is the locked box and $d$ is the key such that (1) $c$ reveals no information about $m$, but (2) together $(c, d)$ reveal $m$, and it is infeasible to find $d'$ such that $(c, d')$ reveals $m' \neq m$. This is defined formally next.

[1] This can be contrasted with a more general *protocols* when the sender and the recipient can send messages to each other in multiple rounds.

## 1.2 Definition

DEFINITION 1 [Commitment Scheme] A (non-interactive) Commitment Scheme (for a message space $M$) is a triple $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ such that:

(a) $\mathsf{CK} \leftarrow \mathsf{Setup}(1^k)$ generates the public commitment key.

(b) for any $m \in M$, $(c, d) \leftarrow \mathsf{Commit}_{\mathsf{CK}}(m)$ is the commitment/opening pair for $m$. $c = c(m)$ serves as the commitment value, and $d = d(m)$ as the opening value. We often omit mentioning the public key $\mathsf{CK}$ when it is clear from the context.

(c) $\mathsf{Open}_{\mathsf{CK}}(c, d) \rightarrow \tilde{m} \in M \cup \{\perp\}$, where $\perp$ is returned if $c$ is not a valid commitment to any message. We often omit mentioning the public key $\mathsf{CK}$ when it is clear from the context.

(d) Correctness: for any $m \in M$, $\mathsf{Open}_{\mathsf{CK}}(\mathsf{Commit}_{\mathsf{CK}}(m)) = m$

$\diamondsuit$

Here is how a commitment scheme is used. If Bob wants to commit a value $m$ to Alice (using the commitment key $\mathsf{CK}$ which we don't explicitly mention below), he first generates the pair $(c, d) \leftarrow \mathsf{Commit}(m)$, and sends $c$ to Alice. Naturally, this is called the *commit* stage. Later, when he wants to open $m$, he sends $d$ to Alice, who runs $\tilde{m} \leftarrow \mathsf{Open}(c, d)$, and accepts the value $\tilde{m}$ provided that $\tilde{m} \neq \perp$. This is called the *reveal* (or opening) stage. By correctness, $\tilde{m} = m$ if everybody is honest.

**Security.** As we stated informally, we want two *security* properties: (1) $c$ gives Alice no information about $m$, and (d) Bob cannot open $c$ in two different ways. The properties stated above are called *hiding* and *binding*.

1. **Hiding**. It is computationally hard for any adversary $A$ to generate two messages $m_0, m_1 \in M$ such that $A$ can distinguish between their corresponding locked boxes $c_0, c_1$. That is, $c(m)$ reveals no information about $m$. Formally, for any PPT $A = (A_1, A_2)$ we require:

$$\Pr\left[\, b = \tilde{b} \;\middle|\; \begin{array}{c} \mathsf{CK} \leftarrow \mathsf{Setup}(1^k),\ (m_0, m_1, \alpha) \leftarrow A_1(\mathsf{CK}),\ b \leftarrow_r \{0,1\}, \\ (c, d) \leftarrow \mathsf{Commit}_{\mathsf{CK}}(m_b),\ \tilde{b} \leftarrow A_2(c;\ \alpha) \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(k)$$

   We write $c(m_0) \approx c(m_1)$, for any $(m_0, m_1)$ chosen by $A$.

2. **Binding**. It is computationally hard for the adversary $A$ to come up with a triple $(c, d, d')$, referred to as a *collision*, such that $(c, d)$ and $(c, d')$ are valid commitments for $m$ and $m'$ and $m \neq m'$. Formally, for any PPT $A$ we require:

$$\Pr\left[\, \begin{array}{l} m \neq m' \wedge \\ m, m' \neq \perp \end{array} \;\middle|\; \begin{array}{c} \mathsf{CK} \leftarrow \mathsf{Setup}(1^k),\ (c, d, d') \leftarrow A(\mathsf{CK}) \\ m \leftarrow \mathsf{Open}_{\mathsf{CK}}(c, d),\ m' \leftarrow \mathsf{Open}_{\mathsf{CK}}(c, d') \end{array} \right] \leq \mathsf{negl}(k)$$

### 1.3 Comments

**Commitment and Encryption.** The hiding property of commitments is exactly the same as for (CPA-secure) public-key encryption: namely, $c(m_0) \approx c(m_1)$, for any $m_0$ and $m_1$. The binding property also seems similar. For commitments it says that every $c$ can be opened in at most one way. Translated to encryption, it says that any encryption can be decrypted in at most one way. The types of encryptions we studied in this class indeed satisfy this property. Indeed, in our definitions Alice — the owner of the secret key — can *always* correctly decrypt any message sent to her using her public key $PK$): for any $m$, $SK$ and $PK$, $D_{SK}(E_{PK}(m)) = m$. Thus, if there was a ciphertext $c$ and two secret keys $SK_0$ and $SK_1$ — both corresponding to $PK$ — such that $m_0 = D_{SK_1}(c) \neq D_{SK_2}(c) = m_1$, then the sender Bob of $m_0$ cannot be sure that $E_{PK}(m_0)$ will not decrypt to $m_1$. This is because it could happen that Alice's secret key corresponding to $PK$ is $SK_1$, and Bob was unlucky to generate $E_{PK}(m_0) = c$. Hence, the type of encryption we studied so far is always binding. Not surprisingly, it is called a *committing* encryption.[2] Summarizing our comparison so far, we conclude

**Lemma 1** *A committing encryption implies a secure commitment scheme.*

**Proof:** Assume $\mathcal{E} = (G, E, D)$ that is a CPA-secure PKE. We want to define a secure commitment scheme $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$. This seems straightforward, since we can let the encryption $E_{PK}(m)$ be our commitment, and the secret key $SK$ be a "universal" opening. There is only one minor difficulty in the this: where do we store the secret key $SK$? Certainly, we cannot store it as part of the commitment key (why?). But then where do we get it in order to open the commitment then? The answer is simple. We don't need the secret key: we can open $c$ by demonstrating the randomness $r$ used for encryption! We get the following scheme:

(i) Let $\mathsf{Setup}(1^k)$ output $\mathsf{CK} = PK$, where $(PK, SK) \leftarrow G(1^k)$.

(ii) Let $(c, (m;\ r)) \leftarrow \mathsf{Commit}(m;\ r)$, where $r$ is chosen at random and $c = E_{PK}(m;\ r)$.

(iii) Let $\tilde{m} \leftarrow \mathsf{Open}(c, (m;\ r))$, where $\tilde{m} = m$ if $c = E_{PK}(m;\ r)$ and $\tilde{m} = \perp$ otherwise.

The fact that this is a secure commitment follows easily from the discussion above (and the fact that $\mathcal{E}$ is committing). □

If we examine the proof above, we see that the secret key was never used! This exemplifies the crucial difference between commitment and encryption: encryption requires also the ability to decrypt based on $c$ and "universal" secret key (*independent of the message*), while commitment allows to "decrypt" with *message-dependent* "secret key" $d$. In particular, almost always $d$ contains the message $m$ in the clear. In fact, we can assume without loss of generality that $d = (m, r)$, and $\mathsf{Open}(c;\ (m, r))$ simply checks if $c = \mathsf{Commit}(m;\ r)$ (notice, the proof above followed this format), and outputs $m$ if the check succeeds.

[2] In turns out that often people allow a more relaxed notion of encryption, where one can have a negligible probability of decryption errors. We will not talk about this further in this introductory course, but remark that in certain applications, like secure multi-party computation and electronic voting, such "non-committing" encryption is extremely useful.

This makes the design of commitment schemes much easier that that of public-key (committing) encryption. In particular, we will shortly see that the construction in Lemma 1 is an "overkill": much simpler construction exist. However, we will also see that the close similarity between commitment and encryption will make the former inherit some properties of the latter.

**Commitment and Signatures.** There is also a much less obvious similarity between commitments and (public-key) signatures. The bonding property of a commitment scheme in some sense implies that the commitment $c$ "validates" $m$, since $c$ cannot be open to a different value. However, this similarity is a bit far fetched. First, the "signature" $c = c(m)$ is not publicly verifiable. Namely, it requires the "signer" to release $d$. In fact, the hiding property even implies that $c$ does not even reveal the message "signed"! Also, the security of signatures says that it is hard to forge a "new" signature. Here everyone can forge signature, since the commitment key is public. Instead, it only says that it is hard to forge a "signature" of a message which is equal to a "signature" of a different message. However, the fact that both signatures and commitments cannot be shorter than the message makes collision-resistant hash function very useful for both, as we shall see.

**Who runs the setup algorithm $\mathsf{Setup}(1^k)$?** In our definition $\mathsf{CK}$ is public information. However it is not clear who generates it: the sender or the receiver. The problem is that if a single party generates it, it can potentially generate it way that benefits this party. For example, the recipient Alice might generate a $\mathsf{CK}$ that would always allow her to see inside the locked box $c$ and determine (partial information about) $m$; thus, breaking the hiding property. On the other hand, the sender Bob might generate a $\mathsf{CK}$ that would allow him to generate different values $d$ that would open the locked box $c$ in different ways; thus, breaking the binding property.

It turns out this question is non-trivial. There are several answers. For the simplest answer, we can assume it is done by a trusted third party, and then such key can be subsequently used by any pair of (possibly untrusting) players. Such commitment scheme are said to have "public parameters" (initialized by the trusted party). Of course, we often would like to avoid this assumption. For another answer, in many of commitment schemes (and most of the ones we present) it turns out that it is actually safe to let one specific party (either sender or the recipient depending on the protocol) to run the key generation, and simply announce the commitment key. For example, with committing encryption of Lemma 1 we can let the sender generate this key (but cannot let the recipient do it; why?). More generally, when the scheme is *information-theoretically binding* (i.e., the message is hidden only computationally, but in theory is embedded into $c$) it is often the case that *any* commitment key gives binding. Thus, the sender cannot choose a bad "binding key", and it is in his interests to choose a good "hiding key". Similarly, when the scheme is *information-theoretically hiding* (i.e., the message is independent from $c$, but it is computationally hard to break the binding property) it is often the case that *any* commitment key gives hiding (or it is possible to give a certificate that the key is "good hiding"). Thus, the recipient cannot choose a bad "hiding key", and it is in his interests to choose a good "binding key". In such cases we can let the recipient choose the key. Notice, however, if the recipient chooses a new key for every message, the commitment scheme becomes *interactive*. And this is the basis

for the third general answer. Namely, in some cases, the sender and the recipient choose the key (or even perform the whole commitment stage) jointly, by running some *interactive protocol*.

To summarize, the solution depends on the scheme in question. For simplicity, we assume that the key is generated correctly by a trusted third party.

**Asymmetry.** Finally, a commitment scheme is slightly unfair to the recipient because even though the sender commits to a value when sending $c$, the recipient has no way of knowing what value the sender has committed to until the sender sends $d$. Therefore, the sender can simply refuse to send the recipient $d$. It turns out, such asymmetry is inevitable in two party protocols: one party always has an advantage in a sense of aborting the protocol before the other party learned its output.[3]

# 2 Examples of Commitment Schemes

We already gave an example in Lemma 1 based on any committing encryption. As we mentioned, this scheme is a bit of an "overkill". We now give simpler constructions.

## 2.1 Commitment using PRG

### 2.1.1 One-bit Scheme

Given $G$ that is a $\mathsf{PRG} : \{0,1\}^k \to \{0,1\}^{3k}$, define $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ for $M = \{0,1\}$ such that:

(i) $R \leftarrow \mathsf{Setup}(1^k)$, where $R \leftarrow^r \{0,1\}^{3k}$.

(ii) $(c,\ (s,b)) \leftarrow \mathsf{Commit}(b)$, where $s \leftarrow^r \{0,1\}^k$ and $c = G(s) \oplus (b \cdot R)$. The notation $b \cdot R$ means $0 \cdot R = 0^{3k}$, and $1 \cdot R = R$. Thus, $c = G(s)$ if $b = 0$, and $c = G(s) \oplus R$ if $b = 1$.

(iii) $\tilde{m} \leftarrow \mathsf{Open}(c,\ (s,b))$, where $\tilde{m} = b$ if $c = G(s) \oplus (b \cdot R)$, and $\tilde{m} = \perp$ otherwise.

### 2.1.2 Security

Hiding is achieved because $c(0) = G(s) \oplus (0 \cdot R) = G(s)$ and $c(1) = G(s) \oplus (1 \cdot R) = G(s) \oplus R \equiv R$ and $G(s) \approx R$ by definition of $G$ being a PRG. Now the question is whether or not the scheme achieves binding. Consider two valid commitments $(c, (s_0, G))$ and $(c, (s_1, G))$ such that $\mathsf{Open}(c, (s_0, G)) = 0$ and $\mathsf{Open}(c, (s_1, G)) = 1$ then $G(s_0) = c$ and $G(s_1) \oplus R = c$. Thus $G(s_0) = G(s_1) \oplus R$ or written in a different way $G(s_0) \oplus G(s_1) = R$. Now there are at most $2^k$ possible values for each $G(s_1)$ and $G(s_2)$ and at most $2^{2k}$ possible values for $G(s_0) \oplus G(s_1)$ while there are $2^{3k}$ possible values for $R$. Thus,

$$Pr_R(\exists\ s_0, s_1 \text{ s.t. } G(s_0) \oplus G(s_1) = R) \leq \frac{2^{2k}}{2^{3k}} = \frac{1}{2^k} = \mathsf{negl}(k)$$

[3] This "advantage" can be made less and less at the expense of increasing the number of rounds, but it will not concern us.

The probability that $s_0$ and $s_1$ that would produce a collision with a random $R$ even exist is negligible, so the scheme achieves the binding property (information-theoretically).

Notice, this is an example of a scheme, where it is safe for the recipient to generate the commitment key $R$.

### 2.1.3 More Bits

The are two ways to extend this scheme to commit to more bits. One is bit-by-bit composition (see Lemma 2), which would make the sender commit to each bit individually. It turns out we can do better by directly extending the scheme above. For concreteness, assume the message space is $M = \{0,1\}^k$ (the method easily extends to any polynomial message size). Now, consider the finite field $F$ of cardinality $2^{5k}$. The elements of this field are naturally represented as $5k$-bit string. Moreover, both the addition and the subtraction in such representation coincide with the XOR operation $\oplus$ (because the field has characteristic 2). Let $\cdot$ now denote multiplication, and interpret every string $m \in M = \{0,1\}^k$ as $0^{4k} \circ m \in \{0,1\}^{5k}$, so we can view $m$ as a member of $F$. Now, we directly extend our scheme:

(i) $R \leftarrow \mathsf{Setup}(1^k)$, where $R \leftarrow^r \{0,1\}^{5k}$.

(ii) $(c,\ (s,m)) \leftarrow \mathsf{Commit}(m)$, where $s \leftarrow^r \{0,1\}^k$, $c = G(s) \oplus (m \cdot R)$ and addition and multiplication are done in $F$.

(iii) $\tilde{m} \leftarrow \mathsf{Open}(c,\ (s,m))$, where $\tilde{m} = m$ if $c = G(s) \oplus (m \cdot R)$, and $\tilde{m} = \perp$ otherwise.

The hiding is proven as before, since $G(s)$ plus any fixed string looks pseudorandom. As for binding, in order for $G(s_0) \oplus (m_0 \cdot R) = G(s_1) \oplus (m_1 \cdot R)$, where $m_0 \neq m_1$, we must have

$$R = (G(s_0) \oplus G(s_1)) \cdot (m_0 \oplus m_1)^{-1}$$

where the inverse is taken in our field $F$. There are at most $2^{4k}$ values for the quantity $(G(s_0) \oplus G(s_1)) \cdot (m_0 \oplus m_1)^{-1}$, so a random $R \in \{0,1\}^{5k}$ can be of the above form with probability at most $2^{4k}/2^{5k} = 2^{-k} = \mathsf{negl}(k)$, as before.

## 2.2 One-bit Commitment using OWP

### 2.2.1 The Scheme

Given $f$ that is a OWP and $h$ that is a hardcore bit for $f$, define $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ for $M = \{0,1\}$ as follows:

(i) $\mathsf{Setup}(1^k)$ outputs the descryption of $f$ and $h$ (in case those are chosen from a family of OWP's).

(ii) $(c,x) \leftarrow \mathsf{Commit}(b)$ where $x \leftarrow^r \{0,1\}^k$ subject to $h(x) = b$, and $c = f(x)$.

(iii) $\tilde{m} \leftarrow \mathsf{Open}(c,x)$, where $\tilde{m} = h(x)$ if $c = f(x)$ and $\tilde{m} = \perp$ otherwise.

In other words, we use the value $f(x)$ to commit to its hardcore bit $h(x)$.

### 2.2.2 Security

Hiding is achieved because determining $b$ from $c(b)$ for a random $b$ is equivalent to determing $h(x)$ from $f(x)$ for a random $x$. Since $h$ is a hardcore bit for $f$, no adversary can determine $b$ from $c(b)$ and equivalently distinguish between $c(0)$ and $c(1)$ with greater than $1/2+\mathsf{negl}(k)$ probability. Binding is achieved information-theoretically, because $f$ is a permutation, so $c = f(x)$ uniquely determines $x$, and thus $b = h(x)$.

Notice, in this scheme either player can typically run the setup algorithm. The disadvantage of the scheme is that it only allows one to commit to one bit. If several bits of $f$ are simultaneously hardcore, we can use this scheme to commit to more bits, but one typically does not use this scheme for committing to more than one (or very few) bits. Instead, schemes given below are used.

## 2.3 Commitment using CRHF

Assume $\mathcal{H}$ is a collision-resistant family. Intuitively, it is very easy to achieve (computational) binding using a random $h \in \mathcal{H}$, since $h(x)$ commits one to the value of $x$. Unfortunately, $h(x)$ need not (and actually does not) hide all partial information about $x$, so we need to do something more complicated to achieve hiding.

### 2.3.1 The Scheme

So assume $\mathcal{H}$ that is a CRHF from $L$ to $\ell$ bits, let $M = \{0,1\}^n$ and assume $L \gg \ell + n$ (the reason for this will be given later later). Finally, let $\mathcal{U}$ be a family of perfect universal hash functions from $L$ to $n$ bits.[4] Then we define the commitment scheme $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ for $M = \{0,1\}^n$ as follows:

(i) $h \leftarrow \mathsf{Setup}(1^k)$, where $h \leftarrow^r \mathcal{H}$.

(ii) $(c,(u,x)) \leftarrow \mathsf{Commit}(m)$, where $x \leftarrow^r \{0,1\}^k$, $c = (u, h(x))$ and $u$ is a universal hash function chosen fro $\mathcal{U}$ at random subject to $u(x) = m$.

(iii) $\tilde{m} \leftarrow \mathsf{Open}(c,(u,x))$, where $\tilde{m} = u(x)$ if $c = (u, h(x))$ and $\tilde{m} = \perp$ otherwise.

### 2.3.2 Security

Biding is achieved because a collision — necessarily of the form $c = (u,y)$, $d = (u,x)$, $d' = (u,x')$ — implies that $h(x) = h(x') = y$, and since $h$ is chosen at random from CRHF family, the adversary is "forced" to use $x = x'$, but then we get $m = u(x) = u(x') = m'$, so the messages are not distinct. Hiding (which is information-theoretic, up to a negligible statistical advantage) is much more difficult to prove. We will not do it, but notice that this is where the condition $L \gg \ell + n$. Intuitively, $h(x)$ reveals $\ell$ out of $L$ bits of information about randomly chosen $x$. Then, universally of $\mathcal{U}$ and the fact that $L - \ell \gg n$ imply that the choice of $u$ — even subject to $u(x) = m$ — still leaves the distribution of $u$ look "almost uniform" to the adversary, *independent* of $m$ (this is the hard part). Thus, irrespective of

[4] This means that for any $x_0, x_1 \in \{0,1\}^L$, $m_0, m_1 \in \{0,1\}^n$ we have $\Pr_u(u(x_0) = m_0 \wedge u(x_1) = m_1) = \Pr_u(u(x_0) = m_0) \cdot \Pr_u(u(x_1) = m_1)$.

what the message $m \in \{0,1\}^n$ is being committed, the adversary sees a randomly looking $u$ and the value $h(x)$, where both $h$ and $x$ where random and independent of $m$.

We notice that the size of commitment for $n$-bit message is $O(L) \gg n$ (since $u$ takes $O(L)$ bits to represent). We will see later how CRHF's can also be used to decrease the commitment size to $O(\ell)$, which could be much less than $n$.

## 2.4 Using Random Oracle

We leave this an a simple exercise to show that *optimal*[5] commitment schemes are completely trivial to build in the random oracle model (why?).

## 2.5 Pedersen commitment (using discrete log)

Finally, we give an example of a commitment scheme based on a specific number theoretic assumption – the discrete log assumption. The scheme is called Pedersen commitment.

### 2.5.1 One-bit Scheme

Define $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ such that:

(i) $(p, g, y) \leftarrow \mathsf{Setup}(1^k)$ where $p$ is a prime, $y$ is a randomly chosen element of $\mathbb{Z}_p^*$, and $g$ is a randomly chosen generator of $\mathbb{Z}_p^*$.

(ii) $(c, (r, b)) \leftarrow \mathsf{Commit}(b)$, where $r \leftarrow^r \mathbb{Z}_p^*$ and $c = g^r y^b \bmod p$.

(iii) $\tilde{m} \leftarrow \mathsf{Open}(c, (r, b))$, where $\tilde{m} = b$ if $c = g^r y^b$ and $\tilde{m} = \perp$ otherwise.

### 2.5.2 Security

Hiding is achieved (information-theoretically) because $r$ is randomly chosen from $\mathbb{Z}_p^*$, and therefore both $c(0) = g^r$ and $c(1) = g^r y$ are also random elements of $\mathbb{Z}_p^*$. On the other hand, finding $r_0, r_1$ such that $\mathsf{Open}(c, (r_0, 0)) = 0$ and $\mathsf{Open}(c, (r_1, 1)) = 1$ would require that $g^{r_0} = g^{r_1} y$. Then $y = g^{r_0 - r_1}$, and the adversary would have computed the discrete log of randomly chosen $y$: $DL(y) = (r_0 - r_1) \bmod (p-1)$. Hence, under the assumption that discrete log is computationally hard the binding property is achieved.

### 2.5.3 Commitment for many bits

We extend the commitment scheme above to many bits by using the discrete log assumption over $\mathbb{Z}_p$ where $p = 2q + 1$ is a strong $(k+1)$-bit prime (recall, this means that $p = 2q + 1$ where $q$ is prime). We define $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ over $M = \mathbb{Z}_q$ as follows:

(i) $(p, g, y) \leftarrow \mathsf{Setup}(1^k)$, where $p = 2q + 1$ is a strong $(k+1)$-bit prime , and $g$ is a random generator of $G = QR(\mathbb{Z}_p^*)$, and $y$ is a random element of $G$.

(ii) $(c, (r, m)) \leftarrow \mathsf{Commit}(m)$, where $r \leftarrow^r \mathbb{Z}_q^*$ and $c = g^r y^m \bmod p$.

(iii) $\tilde{m} \leftarrow \mathsf{Open}(c, (r, m))$, where $\tilde{m} = m$ if $c = g^r y^m$, and $\tilde{m} = \perp$ otherwise.

[5] It is easy to see that in an optimal commitment scheme we have $|c| \approx k$ and $|d| \approx |m| + k$, where $k$ is the security parameter.

### 2.5.4 Security

Hiding is achieved as before information-theoretically, because $r$ is chosen at random from $\mathbb{Z}_q^*$, so $g^r y^m$ is random in $G$, irrespective of $m$. On the other hand, finding $(r_0, m_0)$ and $(r_1, m_1)$ such that $m_0 \neq m_1$, $\mathsf{Open}(c, (r_0, m_0)) = m_0$, and $\mathsf{Open}(c, (r_1, m_1)) = m_1$ would require that

$$g^{r_0} y^{m_0} = g^{r_1} y^{m_1} \bmod p$$

Then $g^{r_0 - r_1} = y^{m_1 - m_0} \bmod p$, which implies that

$$y = g^{(r_0 - r_1)\cdot (m_1 - m_0)^{-1} \bmod q} \bmod p$$

Notice, $(m_1 - m_0)^{-1} \bmod q$ exists since $m_0 \neq m_1$ and $q$ is prime. Thus, the adversary would have computed the discrete log of $y$ base $g$: $DL_g(y) = (r_0 - r_1) \cdot (m_1 - m_0)^{-1} \bmod q$. Since $y$ is randomly chosen, this contradicts the discrete log assumption over strong primes, so the binding property is achieved as well.

# 3 Composition Properties of Commitment Schemes

## 3.1 Bit-by-bit Composition (many times usage)

First, we consider the question of whether the same commitment scheme could be securely used multiple times. Equivalently, assuming we have a secure commitment scheme for small message space — for concreteness, $M = \{0, 1\}$ — can we build a secure commitment scheme for larger message space by a simple bit-by-bit composition of the base commitment scheme. As we saw, the answer to this question was positive in case of $\mathsf{CPA}$-secure encryption, but negative for the case of signatures. Luckily, the answer is also positive for commitment schemes. Namely (for simplicity we state the result for $M = \{0, 1\}$, but it clearly holds for any base commitment scheme),

**Lemma 2** *If $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ is a secure commitment scheme for $\{0, 1\}$, then $\mathcal{C}'$, obtained from $\mathcal{C}$ by bit-bit-bit composition for $p(k)$ times, is a secure commitment scheme for $\{0, 1\}^{p(k)}$, for any polynomial $p(k)$. In particular, a given commitment scheme can be securely used for committing to multiple messages.*

**Proof Sketch:** The proof that $\mathcal{C}'$ satisfies the hiding property is the same as for the case of encryption: use the hybrid argument and the fact that $\mathsf{Commit}_{\mathsf{CK}}(\cdot)$ is a public operation. The binding property is also similarly proven using the hybrid argument: finding a collision for distinct $m_0, m_1 \in \{0, 1\}^{p(k)}$ implies finding a 0/1-collision at some position $i \in \{1 \ldots p(k)\}$. □

## 3.2 Hash-then-commit with $\mathsf{CRHF}$'s

Secondly, recall that hashing allowed for very compact signature schemes via "hash-then-sign" paradigm. It turns out that the same can be done for commitment schemes, except it is now called "hash-then-commit" paradigm. In essence, similar to signatures and unlike encryption, we use the fact that commitment does not have to enable one to recover the message; it should only be hard to collide two messages.

**Lemma 3** *If $\mathcal{H}$ is a* CRHF *from $L$ to $\ell$ bits and $\mathcal{C}' = (\mathsf{Setup}', \mathsf{Commit}', \mathsf{Open}')$ is a secure commitment scheme for $\ell$-bit messages, then the commitment scheme $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ defined below is secure for $L$-bit messages:*

*(a)* $\mathsf{CK} = (\mathsf{CK}', h)$, *where* $\mathsf{CK}' \leftarrow \mathsf{Setup}'(1^k)$ *and* $h \leftarrow \mathcal{H}$.

*(b)* $(c', (d', m)) \leftarrow \mathsf{Commit}_{\mathsf{CK}}(m)$, *where* $(c', d') \leftarrow \mathsf{Commit}'_{\mathsf{CK}'}(h(m))$.

*(c)* $\mathsf{Open}_{\mathsf{CK}}(c', (d', m)) = \tilde{m}$, *where* $\tilde{m} = m$ *if* $\mathsf{Open}'_{\mathsf{CK}'}(c', d') = h(m)$, *and else* $\tilde{m} = \perp$.

**Proof Sketch:** The hiding property follows easily from that of $\mathcal{C}'$: $\mathsf{Commit}'(h(m_0)) \approx \mathsf{Commit}'(h(m_1))$. For the binding property, a collision triple $(c', (d'_0, m_0), (d'_1, m_1)$ either implies that $h(m_0) = h(m_1)$ — a collision to $h$ — or that $(c', d'_0, d'_1)$ form a collision for the pair $h(m_0) \neq h(m_1)$. □

In particular, we remark that by combining the hash-then-commit technique with the commitment scheme of Section 2.3, we get extremely compact and efficient commitment schemes based on CRHF's, where the size of the commitment to arbitrarily long messages can be as small as the security parameter.

**Using UOWHF's?** Is is interesting to see if the technique above can work if we replace CRHF's with UOWHF's (picking a fresh $h \in \mathcal{H}$ for every commitment), like we had with digital signatures. A moment reflection shows that the answer is *negative* (why? take a look at the binding property). However, it turns out that UOWHF's can be used, as prescribed above, with a slight relaxation of regular commitment schemes, called *relaxed commitments*. As we will see, this relaxed notion suffices for some important applications of commitment, so we treat it next.

### 3.3 Relaxed Commitments and UOWHF's

We now consider *relaxed* commitment schemes, where the (strict) binding property of regular commitment schemes is replaced by the **Relaxed Binding** property. Informally, having the knowledge of CK, it is computationally hard for the adversary $A$ to come up with a message $m$, such that when $(c, d) \leftarrow \mathsf{Commit}(m)$ is generated, $A(c, d, \mathsf{CK})$ produces, with non-negligible probability, a value $d'$ such that $(c, d')$ is a valid commitment to some $m' \neq m$. Formally, for any PPT $A = (A_1, A_2)$,

$$\Pr\left[\begin{array}{c} m \neq m' \wedge \\ m, m' \neq \perp \end{array} \;\middle|\; \begin{array}{c} \mathsf{CK} \leftarrow \mathsf{Setup}(1^k),\ (m, \alpha) \leftarrow A_1(\mathsf{CK}),\ (c, d) \leftarrow \mathsf{Commit}_{\mathsf{CK}}(m), \\ d' \leftarrow A_2(c, d;\ \alpha),\ m' \leftarrow \mathsf{Open}_{\mathsf{CK}}(c, d') \end{array}\right] \leq \mathsf{negl}(k)$$

Thus, $A$ cannot find a collision using a *randomly generated* $c(m)$, even for $m$ of its choice.

As we shall see, (1) relaxed commitment suffice for some important applications of commitment schemes (see authenticated encryption later), (2) UOWHF's can be used in the "hash-then-commit" paradigm, (3) relaxed commitments could be a bit "easier"[6] to construct than regular ones. We start with point (2).

[6] Obviously, both notions are equivalent to OWF's.

**Lemma 4** *If $\mathcal{H}$ is a* UOWHF *from $L$ to $\ell$ bits with key size $p$ and $\mathcal{C}' = (\mathsf{Setup}', \mathsf{Commit}', \mathsf{Open}')$ is a secure* relaxed *commitment scheme for $\ell$-bit messages, then the* relaxed *commitment scheme $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ defined below is secure for $L$-bit messages:*

*(a)* $\mathsf{CK} = \mathsf{CK}'$, *where* $\mathsf{CK}' \leftarrow \mathsf{Setup}'(1^k)$.

*(b)* $((c', h), (d', m)) \leftarrow \mathsf{Commit}_{\mathsf{CK}}(m)$, *where* $(c', d') \leftarrow \mathsf{Commit}'_{\mathsf{CK}'}(h(m))$ *and* $h \leftarrow \mathcal{H}$.

*(c)* $\mathsf{Open}_{\mathsf{CK}}((c', h), (d', m)) = \tilde{m}$, *where* $\tilde{m} = m$ *if* $\mathsf{Open}'_{\mathsf{CK}'}(c', d') = h(m)$, *and else* $\tilde{m} = \perp$.

**Proof Sketch:** The hiding property follows easily from that of $\mathcal{C}'$: $\mathsf{Commit}'(h(m_0)) \approx \mathsf{Commit}'(h(m_1))$. For the relaxed binding property, since a fresh $h \in \mathcal{H}$ is chosen for every message and is part of the commitment $c = (c', h)$, a collision triple $((c', h), (d'_0, m_0), (d'_1, m_1)$ either implies that $h(m_0) = h(m_1)$ — a collision to $h$ that was chosen *at random and after* $m_0$ — or that $(c', d'_0, d'_1)$ form a collision for the pair $h(m_0) \neq h(m_1)$, where again $c'$ was chosen honestly corresponding to $h(m_0)$. □

The result above is not surprising, since in the relaxed binding property, just like in the security of UOWHF's, the commitment/hash function is chosen honestly after the first message is selected. This finishes point (2) above.

As for point (3), consider the construction of commitments from CRHF's, explained in Section 2.3. We notice that replacing a CRHF by a UOWHF (where a fresh function is chosen per every message) will result in a secure *relaxed* commitment. The proof is left as an exercise. To summarize, the relation between CRHF's and UOWHF is very similar to that between regular and relaxed commitments.

# 4 Applications of Commitment Schemes

## 4.1 Bidding and Auctions

Consider the following example. A potential buyer Bob is happy to buy some item for any price less than the buying price $b$. A potential seller Alice is happy to sell the item for any asking price greater than the asking price $a$. Let us assume that $a \leq b$, but the quantities $a$ and $b$ are initially kept secret by Alice and Bob. Assume Alice and Bob agree on a fair protocol where the item is traded for the average price $p = \frac{a+b}{2}$. One naive protocol would be for Alice tell $a$ to Bob, then for Bob tell $b$ to Alice, and then compute the average $c$. Unfortunately, in this case Bob will certainly report $b' = a$, making $p' = a$ as well. Similarly, if Bob goes first, Alice will report $a' = b$ resulting in $p' = b$. Clearly, what we need is exactly a commitment. First, Alice commits to the value $a$ and tells her commitment to Bob: $c = c(a)$. The hiding property ensures that Bob learns nothing about $a$ from $c$. Then Bob tells the value $b$ to Alice. Then Alice opens $c$ to $a$ by sending the opening information $d = d(a)$. The binding property ensures that she can open it in only one way. Then both players compute $p = \frac{a+b}{2}$.

The example above can be generalized to more complicated situations, like auctions. But the point is clear. First, one party commits to some value, then it learns some other information, after which it opens the committed value. We notice the (necessary) weakness of this: the player can always refuse to open the commitment. In some applications, like the

simple buyer/seller game, this does not create serious problems, but in more complicated examples it could result in some unfairness.

We only remark that one has to be careful in such applications. Aside from the problem of commitment key generation discussed earlier, there is also the problem of *non-malleability*, similar to the case of encryption. We will not deal with it here.

### 4.2 Coin-Flipping

Assume Alice and Bob want to jointly flip a fair coin. For example, they agree that if the coin comes heads, they go to the opera, else they go to the soccer game. Naturally, Bob wants the opera, so he is bound to cheat if Alice asks him to select the (digital) outcome of the coin. Similarly, letting Alice do so will certainly result in a field trip to a soccer stadium. This shows that both Alice and Bob should somehow participate. But what can they do together?

The answer, next best to actually flipping a physical coin, is that they should design an *interactive protocol* where neither player can *influence* the outcome by a non-negligible amount. A first naive attempt is the following: Alice sends Bob a random bit $a \in \{0,1\}$, then Bob tells Alice a random bit $b \in zo$, and they output a joint coin flip $f = a \oplus b$. Clearly, however, in this case we might as well let Bob — the second player to go — select the coin. However, using commitments we can actually fix this protocol.

First, Alice commits to her (supposedly random) bit $a$, and sends $c = c(a)$ to Bob. Bob then sends a (supposedly random) bit $b$ to Alice. Then Alice opens $c$ to $a$ by sending the opening information $d = d(a)$. Finally, both player output the value $f = a \oplus b$.

Showing that the above simple protocol is "good" is actually non-trivial in the following sense: we first need to give a *formal* definition of what a secure coin-flipping protocol is! The latter is indeed tricky. For example, a naive attempt might be to say that both $\Pr(f = 0), \Pr(f = 1) \in [\frac{1}{2} - \mathsf{negl}(k), \frac{1}{2} + \mathsf{negl}(k)]$. However, the protocol above does not (and, in fact, no protocol *can*!) satisfy this strong property. The reason is that Alice, who learns the coin value first, might refuse to open $a$ if the coin flip $f = 0$. This way, Bob will never see $f = 0$. It turns out that more or less the best we can do is that both $\Pr(f = 0), \Pr(f = 1) \leq \frac{1}{2} + \mathsf{negl}(k)$, so no value can be forced with "unreasonable" probability, even though one or both parties can prematurely terminate the protocol, possibly based on the final "ideal" outcome $f$. In our protocol, Alice has this ability, while Bob cannot abort the protocol in a way that is dependent on $f$ (by the hiding property of commitments).

We omit more formal treatment, but mention that the protocol above can be shown to satisfy the formal definition of coin-flipping, as outlined above.

### 4.3 Authenticated Encryption and Relaxed Commitments

We have already earned something about *authenticated encryption* from the homework. While encryption provides privacy against eavesdropping, and signatures/MAC's validate the sender and the integrity of the message, in many situations one wants to achieve both privacy and authenticity *simultaneously*. Authenticated encryption makes sense for both private and public settings. For concreteness, we concentrate below on the public setting. In this case, authenticated encryption is also sometimes called *signcryption*.

We briefly sketch how a commitment scheme $\mathcal{C}$ can be used to achieve efficient secure signcryption — from a secure encryption scheme $E$ and a secure signature scheme $S$ (below, $E(m)$ denotes the encryption of $m$, and $S(m)$ denotes a message/signature pair $(m, \sigma)$, i.e. the signature includes the message signed). The two naive way to build signcryption are those of *sequential composition*. Namely, $E(S(m))$ and $S(E(m))$. Under reasonable definitions, both of these methods indeed yield a secure signcryption. However, they have a potential disadvantage that two expensive public-key operations — signing and encrypting — are done sequentially one after another. Below we show a new method where (usually, quite cheap) commitments allow to perform these operations *in parallel*.

First, two auxiliary lemmas, each being interesting, but "useless" on its own.

**Lemma 5** *$(c, E(d))$, where $(c, d) \leftarrow \mathsf{Commit}(m)$, is a secure encryption scheme if and only if $\mathcal{C}$ satisfies the hiding property of commitment schemes. Decryption is done by first decrypting $d$, and then returning $\mathsf{Open}(c, d)$.*

**Proof Sketch:** We use the hiding property of commitment scheme. Intuitively, $c$ does not revel any information about $m$, so so does $E(d)$, since $E$ is a secure encryption. The converse is clear as well. □

The reason the lemma by itself is "useless" is we might encrypt $m$ directly using a secure encryption $E$: just return $E(m)$.

**Lemma 6** *$(S(c), d)$, where $(c, d) \leftarrow \mathsf{Commit}(m)$, is a secure signature scheme if and only if $\mathcal{C}$ satisfies the relaxed binding property of commitment schemes. Verification is done be verifying the signature of $c$ and checking $\mathsf{Open}(c, d) \neq \perp$.*

**Proof Sketch:** We use the relaxed binding property since the pair $(c, d)$ binds one to the message, so it is hard to reuse a valid signature $(S(c), d)$ corresponding to some $m$ (i.e., $c = c(m)$) to produce a forgery $(S(c), d')$ for $m'$: this will create a collision $(c, d, d')$. Thus, the forger is forced to forge a "new" signature $S(c')$, but this is impossible since $S$ is a secure signature scheme. The converse is simple as well. □

We make two comments here. First, as with the previous lemma, this lemma by itself is useless: one might either sign $m$ directly, or use much cheaper hash-sign-method, if the size of the signature is an issue. Secondly, the lemma above shows that *relaxed* commitments are sufficient for the above applications (so UOWHF's can be used).

Now, we combine the above two lemmas and give the following theorem.

**Theorem 1** *$(S(c), E(d))$, where $(c, d) \leftarrow \mathsf{Commit}(m)$, is a secure signcryption if and only if $\mathcal{C}$ is a secure relaxed commitment scheme (decryption and verification is as in the above lemmas). In particular, it is secure with any regular commitment scheme.*

This result is actually useful, since the expensive public key operations are indeed done in parallel. More optimizations of the above idea are possible (i.e., on-line/off-line signcryption), but we omit the details.

### 4.4 Zero-Knowledge

We will not give any details here, but zero-knowledge proofs allow one to proof the validity of some statement or possession of some secret, without revealing any information beyond a validity of the statement proved, or the possession of the secret in question. For a simple example, one can prove that a number $a \in \mathbb{Z}_n^*$ is a quadratic residue without revealing the square root of $a$! For another example in the second category (proofs of knowledge), one can proof the knowledge of the value $x$ such that $g^x = y \bmod p$ without revealing $x$!

No need to say, the study of zero-knowledge proofs is of fundamental importance in cryptography. It turns out that commitment schemes allow one to prove an amazingly powerful statement (do not worry if this makes no sense now): any language in the (huge) complexity class $NP$ has a (computational) zero-knowledge proof.

### 4.5 Password Authentication and Identification Schemes

Recall the usage of $\mathsf{OWF}$'s for password authentication. The server stores the value $f(x)$ in a public file, and the user authenticates by presenting a value $x$. As an alternative, we can use commitment schemes. Namely, we let $(c, d) \leftarrow \mathsf{Commit}(x)$, store $c$ on the server, and present $d$ during authentication (which succeeds if $\mathsf{Open}(c, d) \neq \perp$). This more complicated schemes has several minor advantages over the scheme with $\mathsf{OWF}$'s: (1) the distribution of $x$ does not have to be uniform; (2) even knowing $x$ does not let one authenticate successfully; (3) the value $c$ reveals no partial information about $x$. In practice, however, the above advantages are not essential, since one still needs to remember the value $d$ (which is longer than $x$, for example).

We remark that password authentication schemes above have a serious weakness in that they trust the server (or assume that nobody snoops during authentication). Specifically, snooping the password $x$ or the opening value $d$ allow the adversary unlimited future access. It turns out that by *combining* either one of the above scheme with an appropriate zero-knowledge proof — where the user proves to the server the *knowledge* of the corresponding authentication information (i.e., $x$ such that $f(x) = y$, or $d$ such that $\mathsf{Open}(c, d) \neq \perp$) *without revealing this information* — allow one to make up a secure *identification scheme*. Specifically, an identification scheme remains secure for the future, even if the adversary manages to listen in during user authentication, and even if it plays the role of the server with the honest user! Intuitively, the only thing such an adversary learns is that the user indeed possesses correct secret information, but this does not help the adversary to impersonate the user, since the adversary new that the user is "legal" to begin with!

We will study this more formally a bit later, after we talk about zero-knowledge in more detail.

### 4.6 Trapdoor Commitments

This is not an application by itself. Rather, it is a *stronger type* of commitment, with one additional property. We will not give more detail now, but remark that trapdoor commitments have found numerous applications including on-line/off-line signatures, chameleon signatures, certified e-mail, zero-knowledge and general multi-party computation.

# Zero-Knowledge Proofs

Yevgeniy Dodis
New York University

Special thanks: Salil Vadhan

# Zero-Knowledge Proofs [GMR85]

- Interactive proofs that reveal nothing other than the validity of assertion being proven

- Central tool in study of cryptographic protocols
  - In particular, central in multi-party computation

- Source of interaction between cryptography & complexity theory

# Outline

- Interactive Proofs
  - Classes NP and IP
- Zero-Knowledge proofs
  - NP has ZK proofs
  - Proofs of Knowledge
  - Honest-Verifier ($\Sigma$-Protocols)
- Composition and Concurrency Issues
  - Universal Composability, Global setup, …

# Proofs

- Prover P wants to prove to Verifier V some statement S is true
- Witness of S: string w s.t. V can check S is true using w
  - Example: S = "Second bit of Dlog(y) is 0", then w = Dlog(y). Test by seeing $w_2$=0 and $g^w = y$
- NP = class of problems where each true statement has a witness, and false statements do not have any witnesses
  - Note, witness might be hard to find, but always easy to check ! (big question: P ≠ NP?)

# NP, formally

- Def: A proof system for a language $L$ is a poly-time algorithm $V$ ("verifier") s.t.
  - Completeness ("true assertions have proofs"):

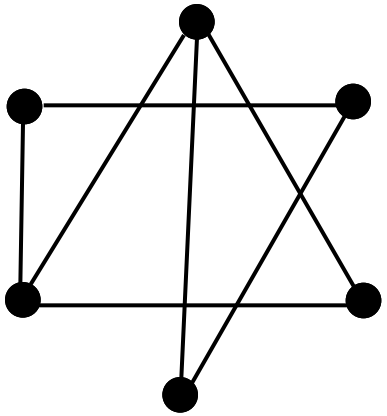
    $x \in L \Rightarrow \exists\, proof$ s.t. $|proof| \leq \text{poly}(|x|)$ and $V(x, proof) = \text{accept}$
  - Soundness ("false assertions have no proofs"):

    $x \notin L \Rightarrow \forall\, proof^*$ $V(x, proof^*) = \text{reject}$

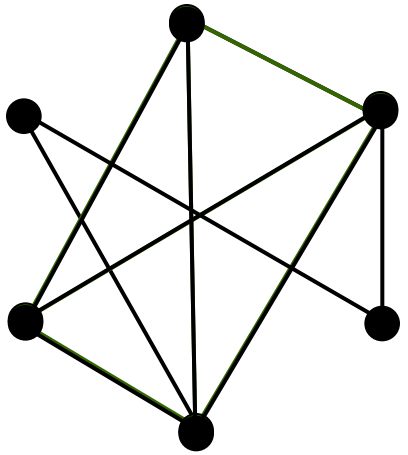
- NP = class of languages w/ proof systems.

# GRAPH 3-COLORING

- Def: A 3-coloring of a graph is an assignment of "colors" in {●,●,●} to vertices s.t. no pair of adjacent vertices are assigned the same color.



a 3-colorable graph



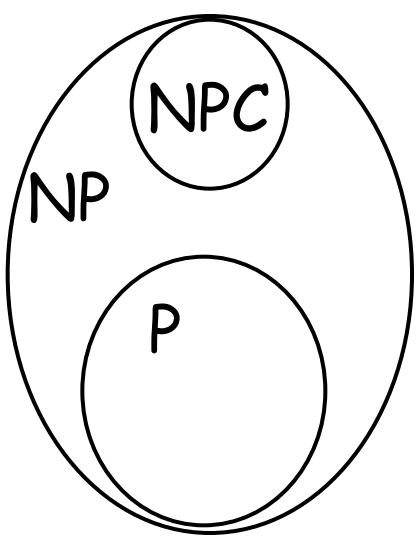
a non-3-colorable graph

- Prop: 3-COL = $\{G : G$ is 3-colorable$\}$ is in NP.

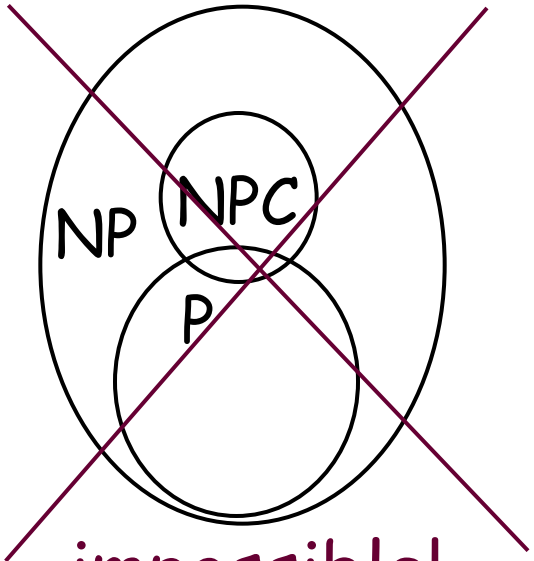
# NP-completeness

- Def: A language L is NP-complete if
  - $L \in NP$
  - Every language in NP "reduces" to L

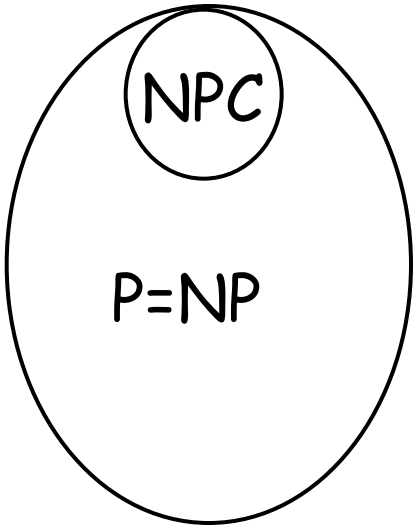
- Thm [C71,L72,K72]: 3-COL is NP-complete.




probable




impossible!




conceivable

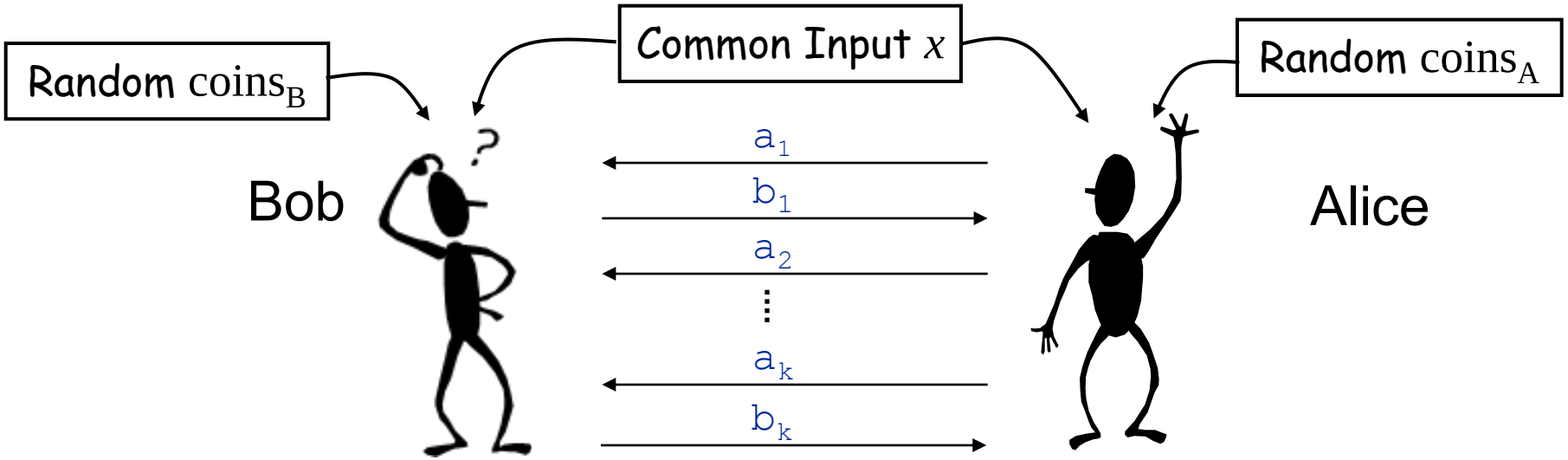
# New Ingredients

- Classical NP proofs inherently non-zero-knowledge.
  - Verifier gains ability to prove $x \in L$ to others.

- Randomization: verifier can "toss coins"
  - Allow verifier to err with small probability

- Interaction: replace static *proof* with dynamic, all-powerful *prover*
  - Will "interact" with verifier and try to "convince" it that assertion is true.

# Interactive Protocols




- Alice, Bob are functions :

(common input, random coins, previous msgs)$\mapsto$(next msg)

- messages $\in \Sigma^* \cup$ {accept, reject, halt}

- Require protocol to be polynomially bounded: lengths of all msgs & # of messages $\leq$ poly($|x|$).

# Interactive Proofs [GMR85,B85]

Def: An interactive proof system for a language $L$ is an interactive protocol $(P,V)$ where

- $V$ is poly-time computable.
- Completeness: If $x \in L$, then
  $V$ accepts in $(P,V)(x)$ with probability 1
- Soundness: If $x \notin L$, then for every $P^*$,
  $V$ accepts in $(P^*,V)(x)$ with probability $\leq 1/2$

Def: IP = { $L$ : $L$ has an interactive proof }

# Comments on Definition

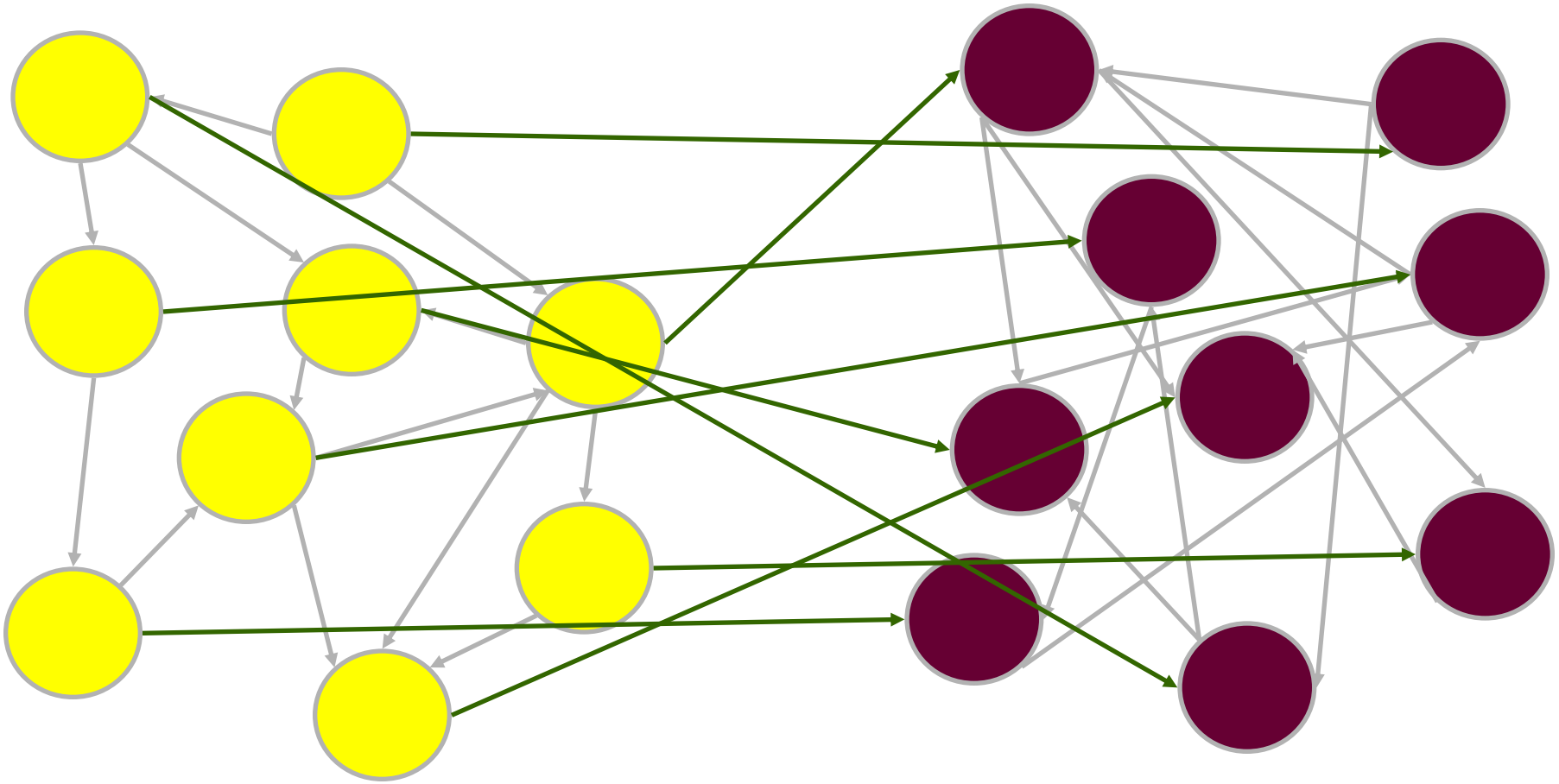
- Asymmetry between "yes" and "no" instances
- Probabilities taken only over coin tosses, not over input.
  - Exercise: if V is deterministic, IP would collapse to NP
- Can reduce error probability (in soundness) to $2^{-1000}$ with $1000$ repetitions.
- Interactive proofs generalize classical proofs: NP$\subseteq$IP.
- IP seems much bigger: IP=PSPACE [LFKN90,S90]

# Example: Graph Isomorphism

- The graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ are called isomorphic (denoted $G_1 \cong G_2$) if there exists a 1-1 and onto mapping $\pi: V_1 \to V_2$ such that $(u,v) \in E_1$ iff $(\pi(u),\pi(v)) \in E_2$.
- A mapping $\pi$ between two isomorphic graphs is called an isomorphism between the graphs.
- If no such mapping exists, the graphs are called non-isomorphic.
- We define the languages
  - $GI=\{(G_1,G_2)$: $G_1$ and $G_2$ are isomorphic$\}$
  - $GNI=\{(G_1,G_2)$: $G_1$ and $G_2$ are non-isomorphic$\}$
- We will use these language in order to demonstrate the power of interactive proofs.

# Isomorphic Graphs

- Take these two graphs
- Although they seem very different, they are in fact isomorphic.



# Observations

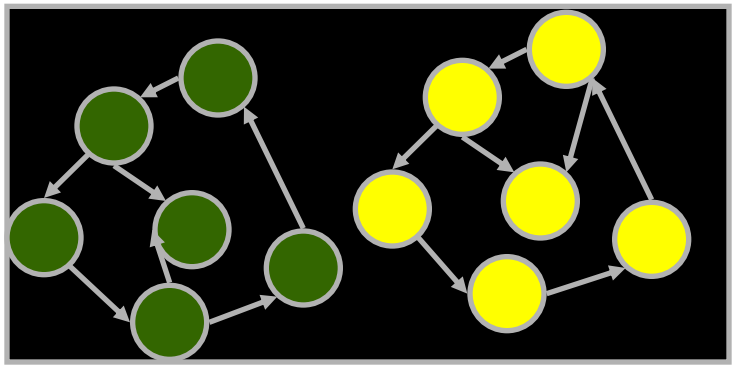
- Clearly, GI is in NP
  - Isomorphism is the witness. Unlikely to be NP-complete
- Interestingly, GNI it is not known to be either in NP, or to be NP-hard (both seem unlikely)
  - Hard to check that no isomorphism exists
- Question 1: how can a Prover convince a Verifier that two graphs are non-isomorphic (without "having" the witness)?
- Question 2: how can a Prover convince a Verifier that two graphs are isomorphic without revealing the isomorphism?

# An Interactive Proof for GNI

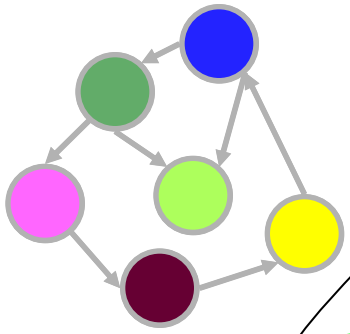
- Common Input: $G_1=(\{1,\ldots,n\},E_1)$ and $G_2=(\{1,\ldots,n\},E_2)$

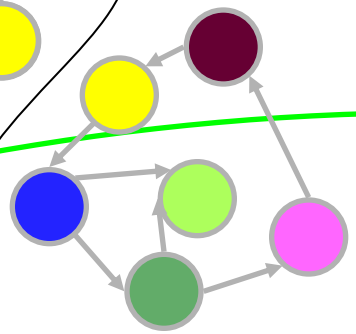
- The Verifier chooses randomly $i$ in $\{1,2\}$ and a permutation $\pi$ of $\{1,\ldots,n\}$.
- Then it applies $\pi$ on the $i$-th graph to get: $H=(\{1,\ldots,n\},\{(\pi(u),\pi(v)):(u,v)\in E_i\})$
- And sends $H$ to the Prover.
- The Prover sends $j\in\{1,2\}$ to the Verifier.
- The Verifier accepts iff $i=j$.

# An Interactive Proof for GNI



The common input





The Prover

- The verifier chooses one of the two graphs randomly.
- The verifier constructs randomly a graph isomorphic to the graph it chose.
- The verifier sends the prover the graph
- The verifier can check the answer easily (The verifier knows which graph was chosen)

- If the two input graphs are truly non-isomorphic, the prover can find which of the two graphs is isomorphic to the graph he received from the verifier, and send it the correct answer.

The Verifier

The 2nd Graph

# The protocol is IP

- Completeness:
  If $G_1$ and $G_2$ are non-isomorphic, the graph the verifier sends is isomorphic to only one out of the two graphs, thus the prover can always send the correct answer.

- Soundness:
  If $G_1$ and $G_2$ are isomorphic, then, since the verifier chooses $i$ randomly, the probability that $j=i$ is at most $\frac{1}{2}$.

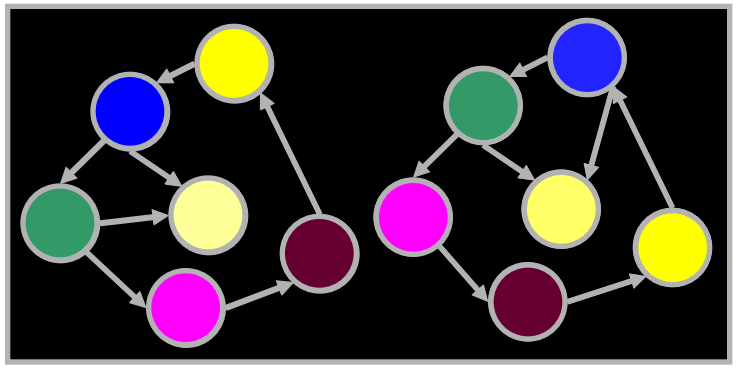
# Is this "ZK"?

- Yes !
  - V knows in advance the response of P…
- Or does he???
  - What if V chose H in a different way?
  - Then he learns which graph H is isomorphic to
- So maybe No? (stay tuned)

# An Interactive Proof for GI

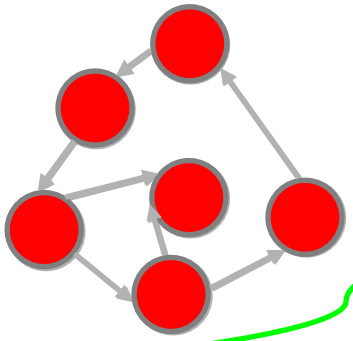
- Common Input: $G_1=(\{1,\ldots,n\},E_1)$ and $G_2=(\{1,\ldots,n\},E_2)$
- Prover knows (can find) permutation $\rho$ from $G_2$ to $G_1$
- The Prover choose a random permutation $\pi$ of $\{1,\ldots,n\}$
- Then it applies $\pi$ on the $G_1$ to get:
  $H=(\{1,\ldots,n\},\{(\pi(u),\pi(v)):(u,v)\in E_1\})$
- And sends $H$ to the Verifier.
- The Verifier sends random $j\in\{1,2\}$ to the Prover.
- If $j=1$, Prover sends $\tau = \pi$ to Verifier, else he sends $\tau = \pi\rho$
- The Verifier checks that $\tau(G_j) = H$
  - If $j=1$, then $\pi(G_1) = H$
  - If $j=2$, then $\pi(\rho(G_2)) = \pi(G_1) = H$

# An Interactive Proof for GI

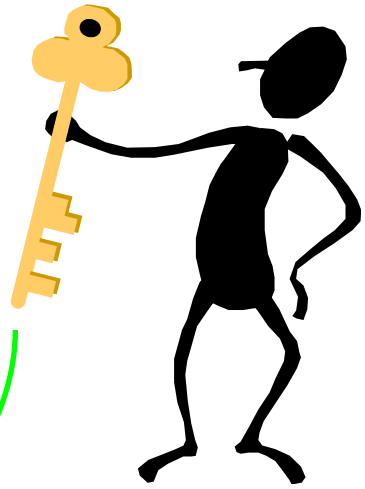


The common input

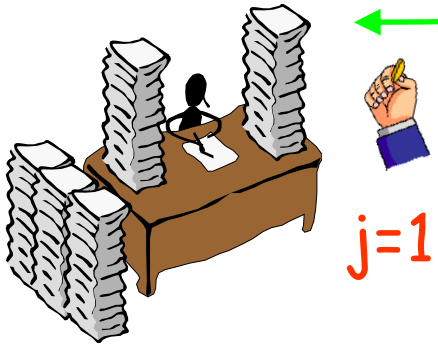
Knows the isomorphism!
Picks random H isomorphic to $G_1$



The Prover

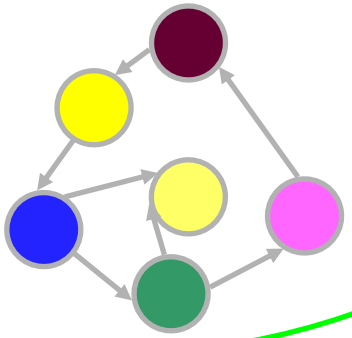


Open isomorphism to $G_1$





j=1



Here you go

The Verifier

And if j was 2, I would have shown this !

# The protocol is IP

- Completeness:
  If $G_1$ and $G_2$ are isomorphic, the graph the prover sends is isomorphic both of the graph, and he can find the isomorphism.

- Soundness:
  If $G_1$ and $G_2$ are non-isomorphic, then any $H$ is not isomorphic to at least one of them, and since the verifier chooses $j$ randomly, he catches the prover with probability at least $\frac{1}{2}$.

# Is this "ZK"?

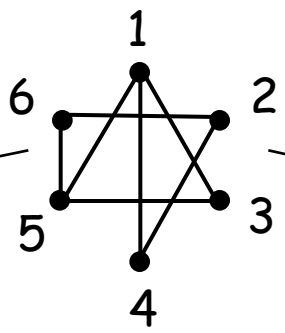
- Yes !
  - He only learned a random graph isomorphic to one of the input graphs
- More formally, given his challenge j, V knows how to prepare H and $\tau$ such that $\tau(G_j) = H$ :
  - Pick $\tau$ at random and set $H = \tau(G_j)$
- Tricky question: but what if Verifier did not choose j at random?
  - Is it ZK? Stay tuned…

# What about all of NP?

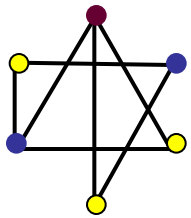
- Previous "ZK protocols" were unconditional
- Can we build unconditional ZK protocol for all of NP?
  - Possible, but very unlikely [GO92,Vad04]
- Nevertheless, possible to build "computationally secure" ZK protocol for all of NP [GMW91] !
  - Efficient prover given the witness
  - Can even extend to all of IP = PSPACE !

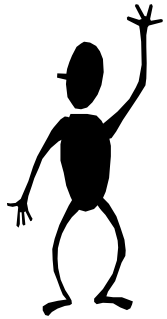
# ZK Proof for 3-COL

poly-time Verifier

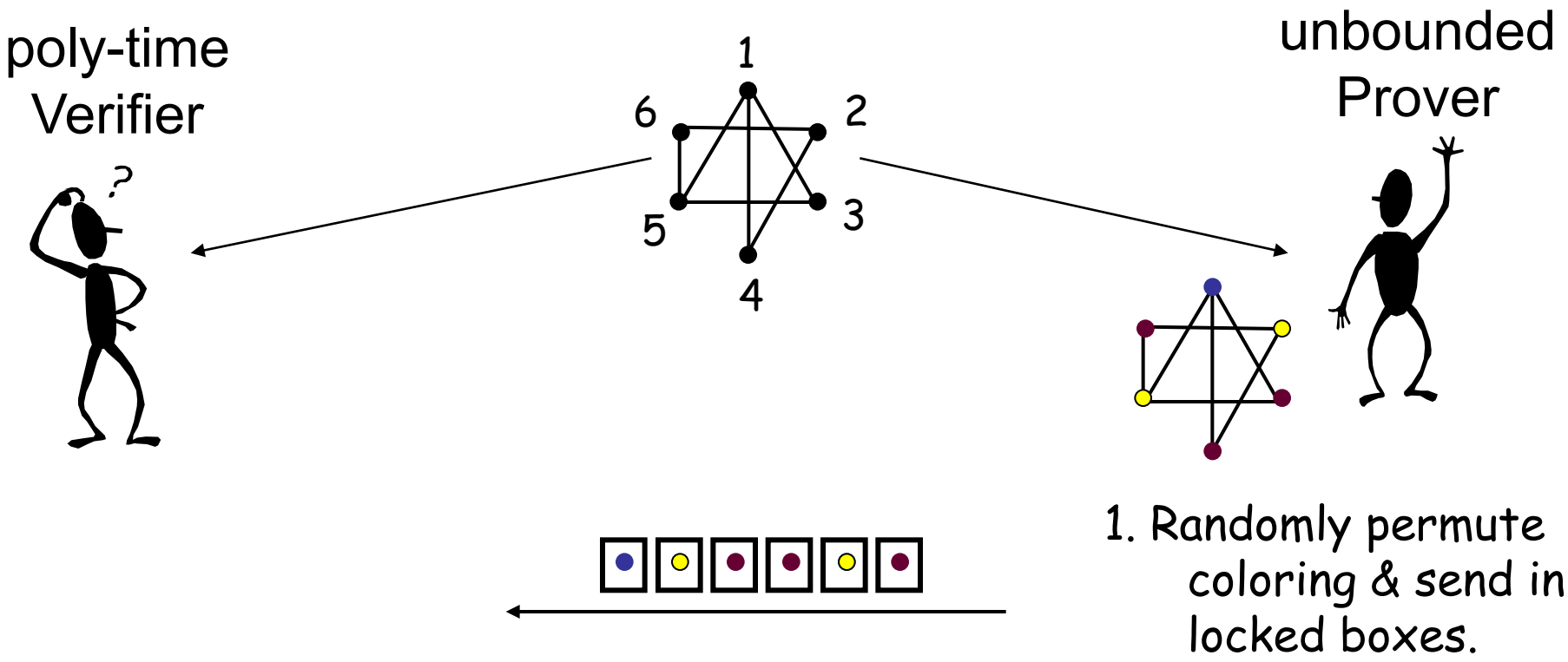

unbounded Prover





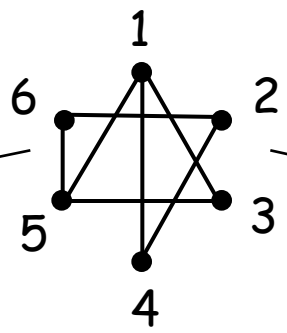
1. Randomly permute coloring & send in locked boxes.

# ZK Proof for 3-COL

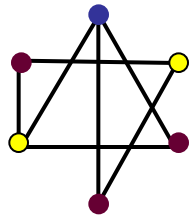


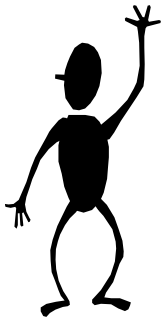
# ZK Proof for 3-COL

poly-time Verifier




unbounded Prover





1. Randomly permute coloring & send in locked boxes.

2. Pick random edge.

(1,4)

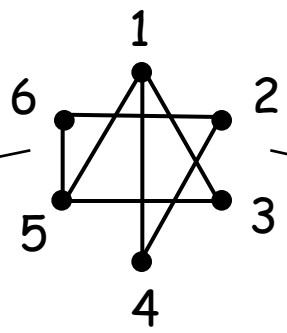
3. Send keys for endpoints.

4. Accept if colors different.

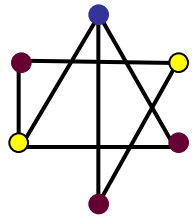
(Perfect) Completeness: graph 3-colorable ⇒ V accepts w.p. 1

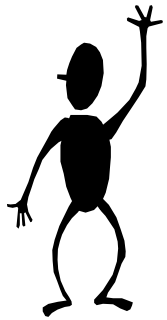
# ZK Proof for 3-COL

poly-time Verifier

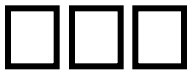



unbounded Prover





1. Randomly permute coloring & send in locked boxes.



2. Pick random edge.

(1,4)

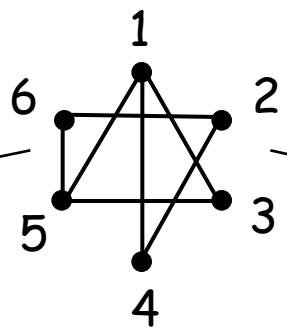
3. Send keys for endpoints.

4. Accept if colors different.

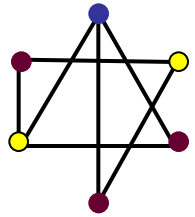
Soundness: graph not 3-colorable $\Rightarrow \forall P^*$ V rejects w.p. $\geq 1/(\#\text{edges})$

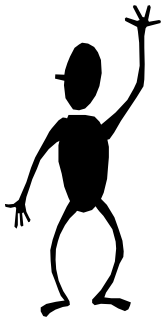
# ZK Proof for 3-COL

poly-time Verifier


unbounded Prover





1. Randomly permute coloring & send in locked boxes.

2. Pick random edge.

(1,4)

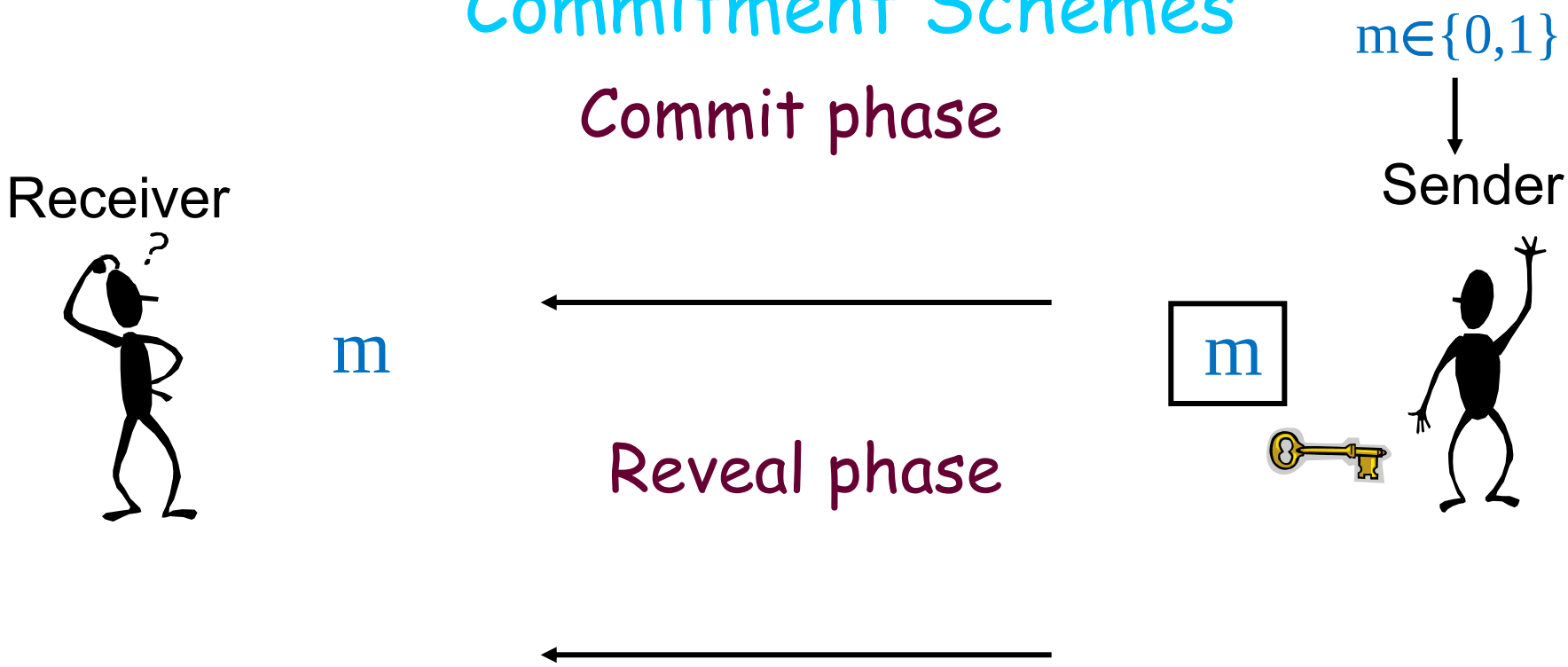
3. Send keys for endpoints.

4. Accept if colors different.

Zero Knowledge: graph 3-colorable ⇒ V sees two random distinct colors

# How to implement boxes?
# Commitment Schemes

# How to implement boxes?
# Commitment Schemes



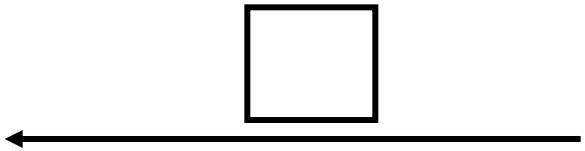
# How to implement boxes?
# Commitment Schemes

$m \in \{0,1\}$

Sender

Receiver

Commit phase



Reveal phase

Hiding property: receiver learns nothing about $m$ from commit phase.

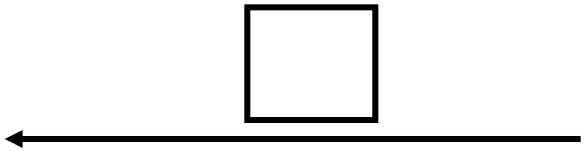
# How to implement boxes?
# Commitment Schemes

$m \in \{0,1\}$

Sender

Receiver

Commit phase

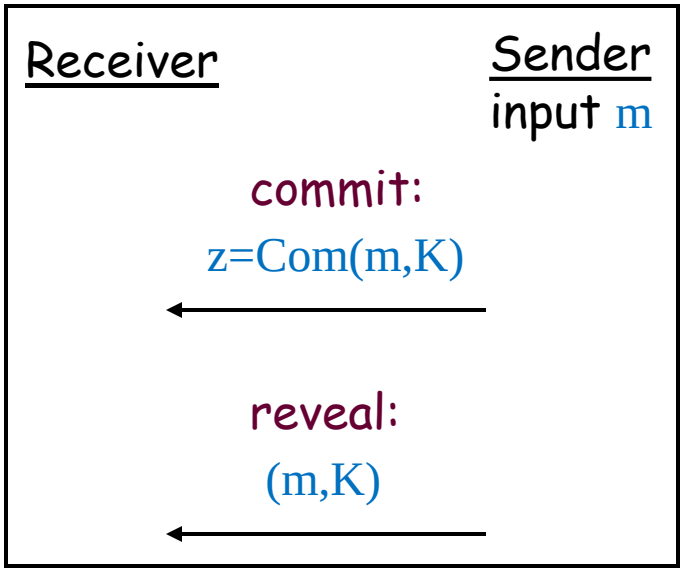


Reveal phase

Binding property: sender cannot change $m$ after commit phase.

# Commitment Schemes

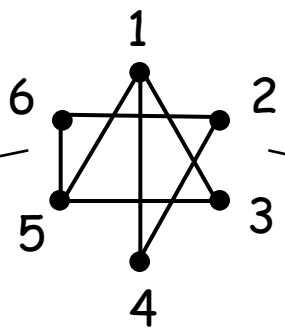
- **Bit-commitment (noninteractive):**
  poly-time computable $Com(m,K)$ s.t.
  - **Hiding:** For random $K_1$ , $K_2$
    $Com(0, K_1)$ & $Com(1, K_2)$
    computationally indistinguishable
    ($\Rightarrow$ zero knowledge)
  - **Binding:** $Com(0, \cdot)$ & $Com(1, \cdot)$ disjoint ($\Rightarrow$ soundness)
    - Collision-resistance enough:
      - Hard to find $K_1$ and $K_2$ s.t. $Com(0, K_1) = Com(1, K_2)$



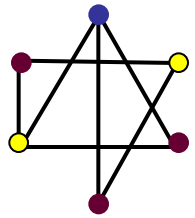

- **Thm [N89,HILL90]:** $\exists$ one-way functions
  $\Rightarrow \exists$ (interactive) bit-commitment schemes.
- Simpler constructions from factoring, discrete log, …
  - Perdersen: $Com(b; K) = g^K h^b \bmod p$
  - Binding: $g^{K_1} = g^{K_2} h \Rightarrow Dlog_g(h) = K_1 - K_2 \bmod q$ ($q$ - group order)

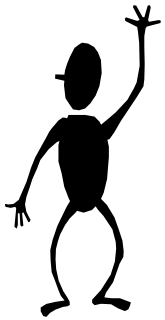
# ZK Proof for 3-COL

poly-time Verifier


unbounded Prover





1. Randomly permute coloring & send in locked boxes.

Com(●)…Com(●)

2. Pick random edge.

(1,4)

3. Send keys for endpoints.

4. Accept if colors different.

$(\bullet,K_1),(\bullet,K_4)$

Zero Knowledge: graph 3-colorable ⇒ V sees two random distinct colors

# Formalizing Zero Knowledge

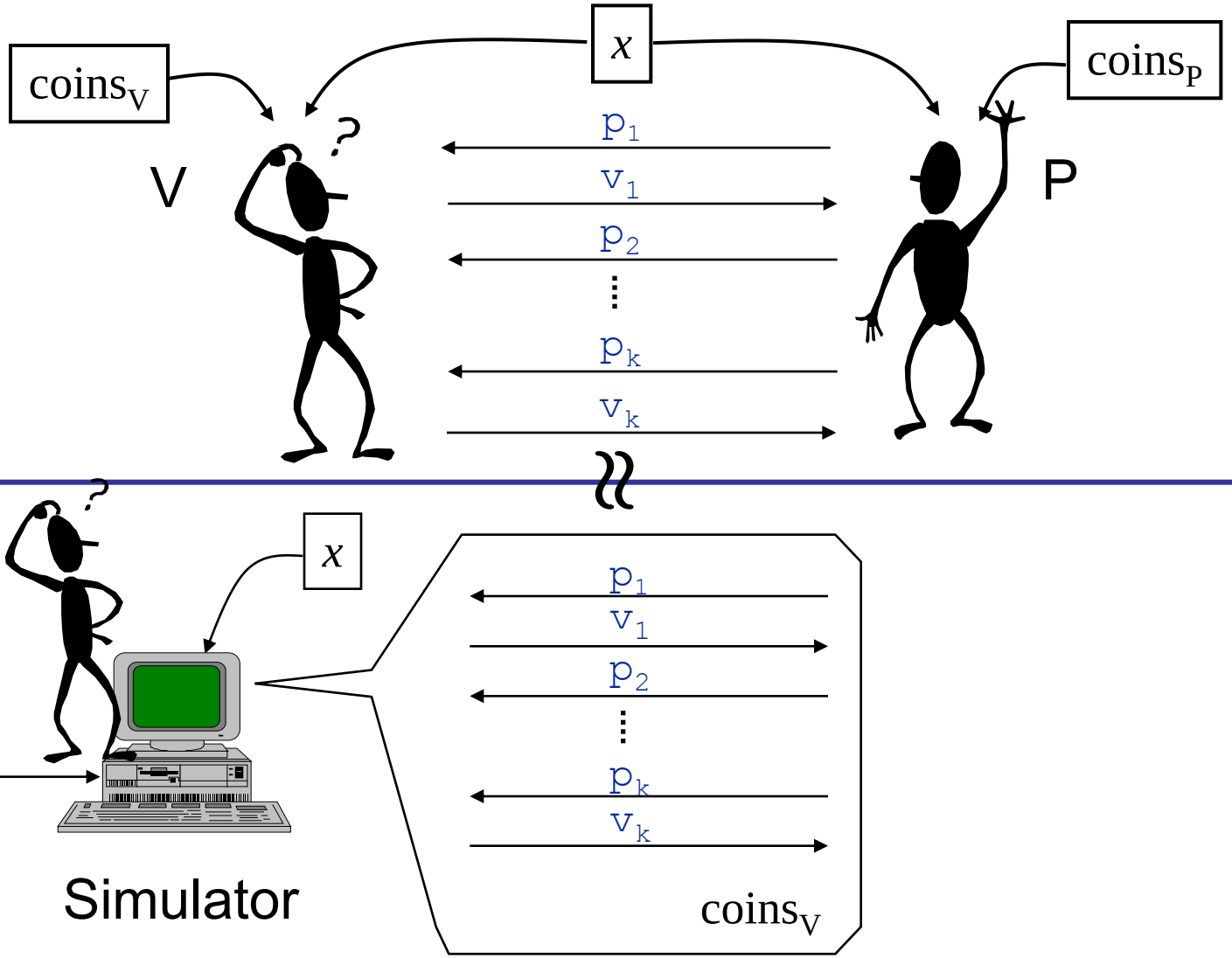
## Simulation Paradigm

*Verifier learns nothing if it could have "simulated" the interaction on its own.*

Def: (P,V) is zero knowledge if

∃ poly-time S s.t. when $x \in L$

S(x) is computationally indistinguishable from (P,V )(x)

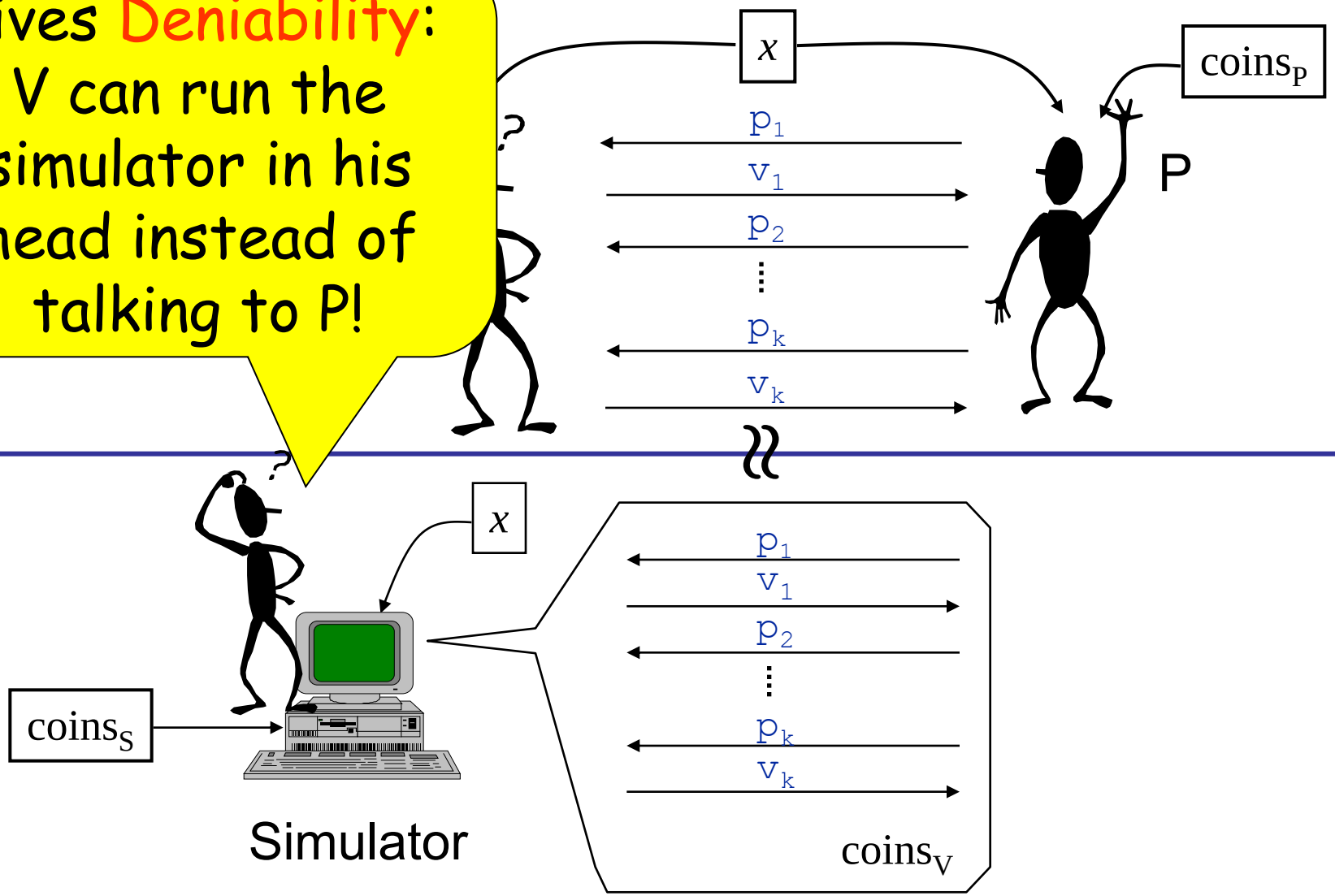


Def: (P,V) is zero knowledge if

∃ poly-time S s.t. when $x \in L$

S(x) is computationally indistinguishable from (P,V )(x)

gives Deniability: V can run the simulator in his head instead of talking to P!



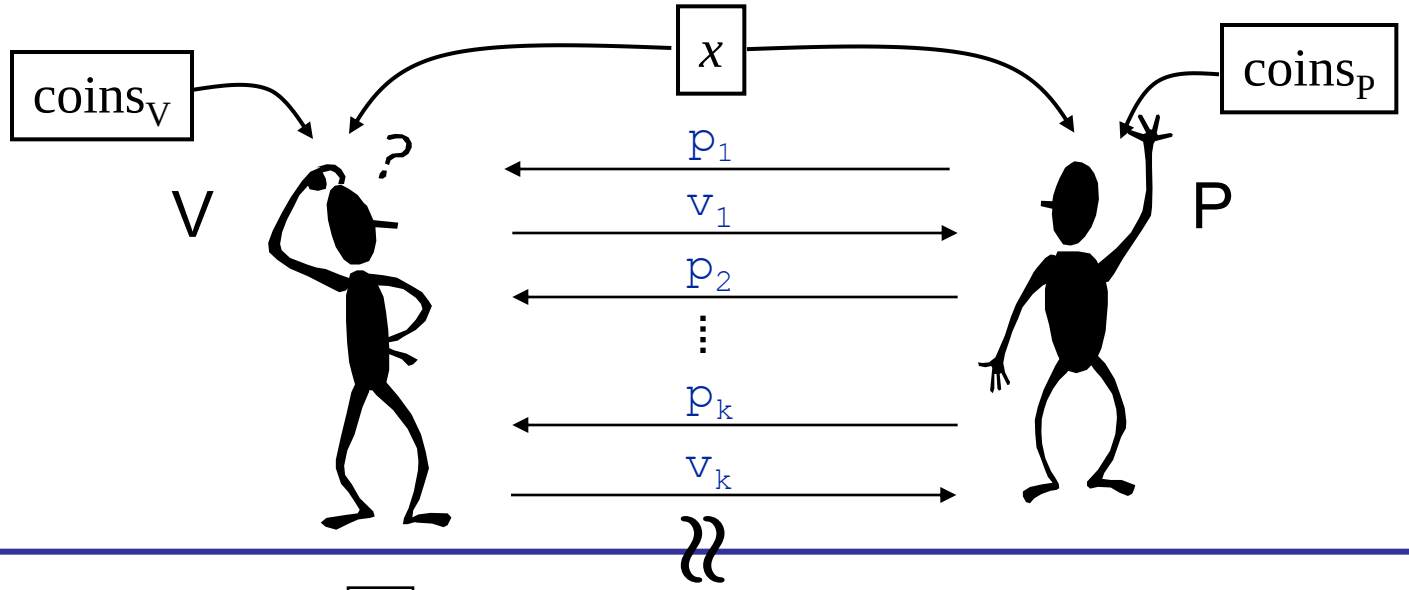
# Is this a good definition?

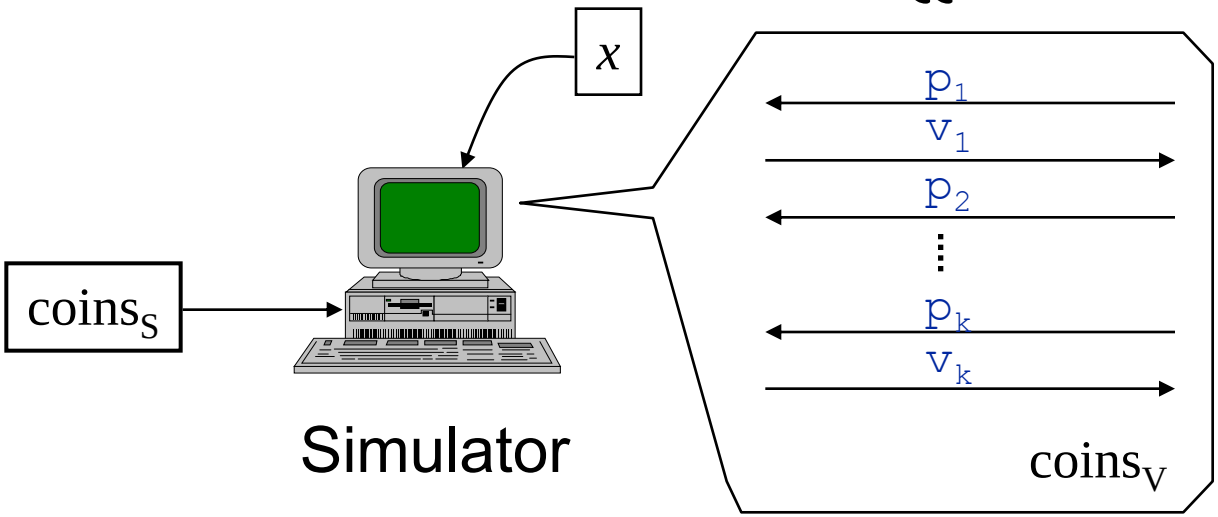
- Yes !
  - GI, GNI, 3-COL protocols had very simple simulators… against honest V
- No !
  - But what if V does not follow the protocol?
- GNI example: learns which graph is isomorphic to a given H, something he maybe did not know!
  - Is it the problem for GI and 3-COL?
  - No, but more complicated analysis (stay tuned)…
- Called honest-verifier ZK (HVZK)
  - Still very useful in many situations (stay tuned)!

Def: (P,V) is honest verifier zero knowledge if

∃ poly-time S s.t. when $x \in L$

S(x) is computationally indistinguishable from (P,V )(x)

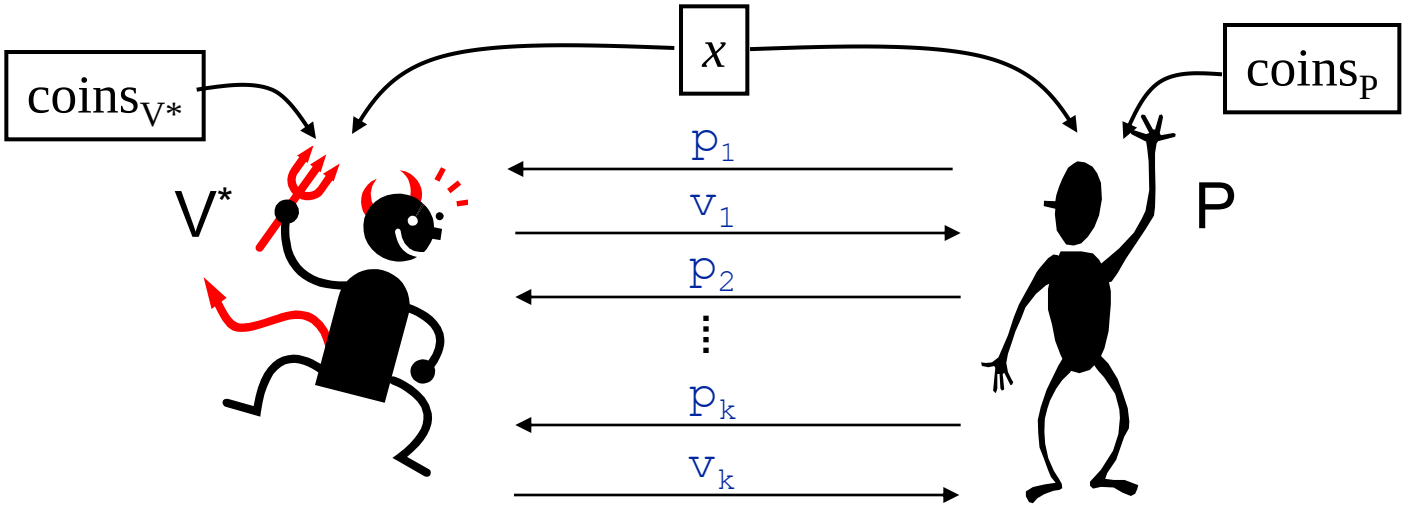


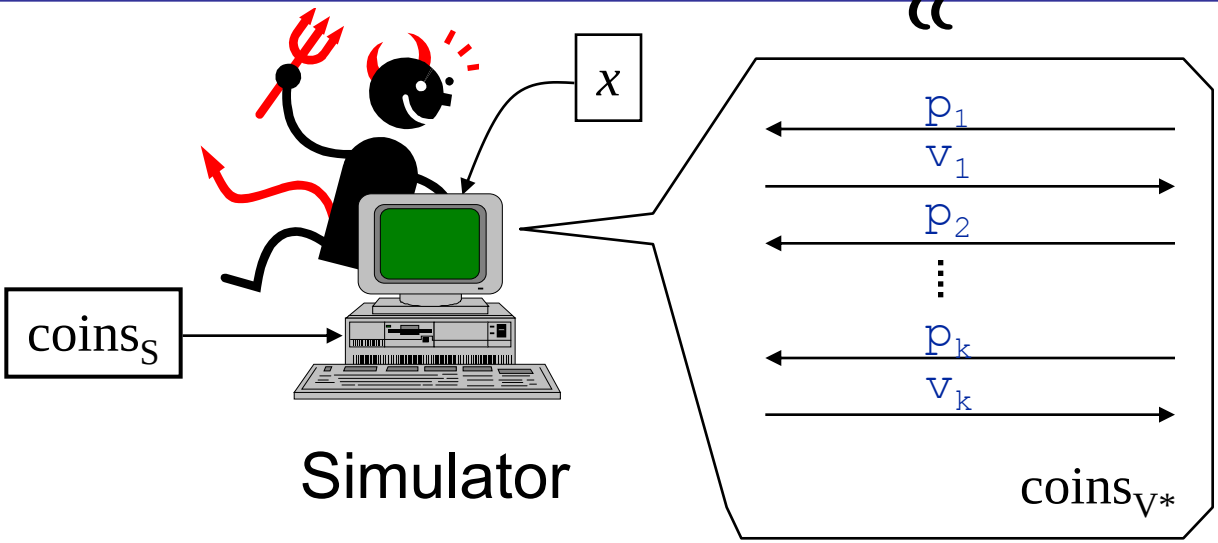



Def: (P,V) is zero knowledge if

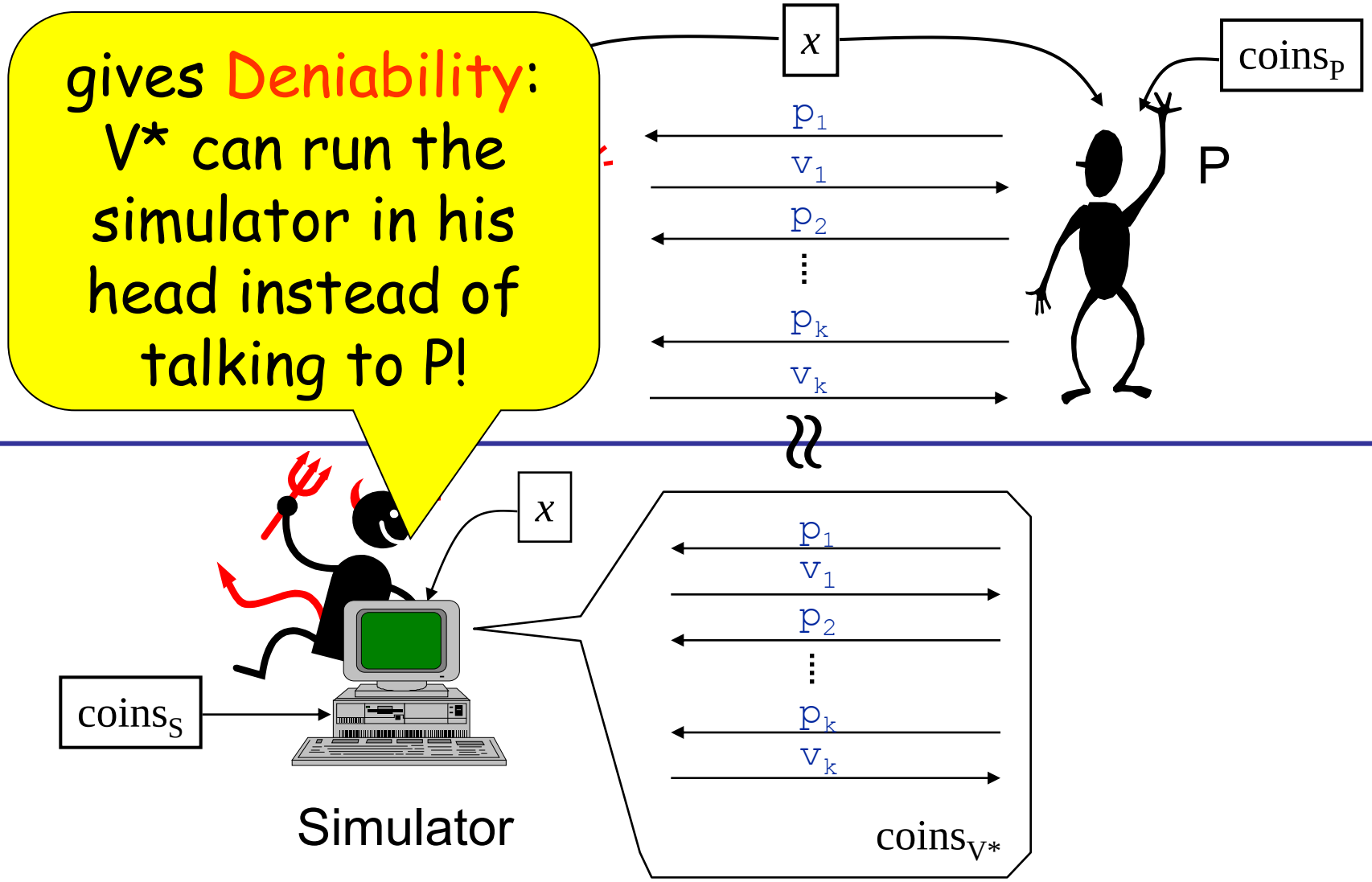
∀ poly-time V* ∃ poly-time S s.t. when $x \in L$

S(x) is computationally indistinguishable from $(P,V^*)(x)$

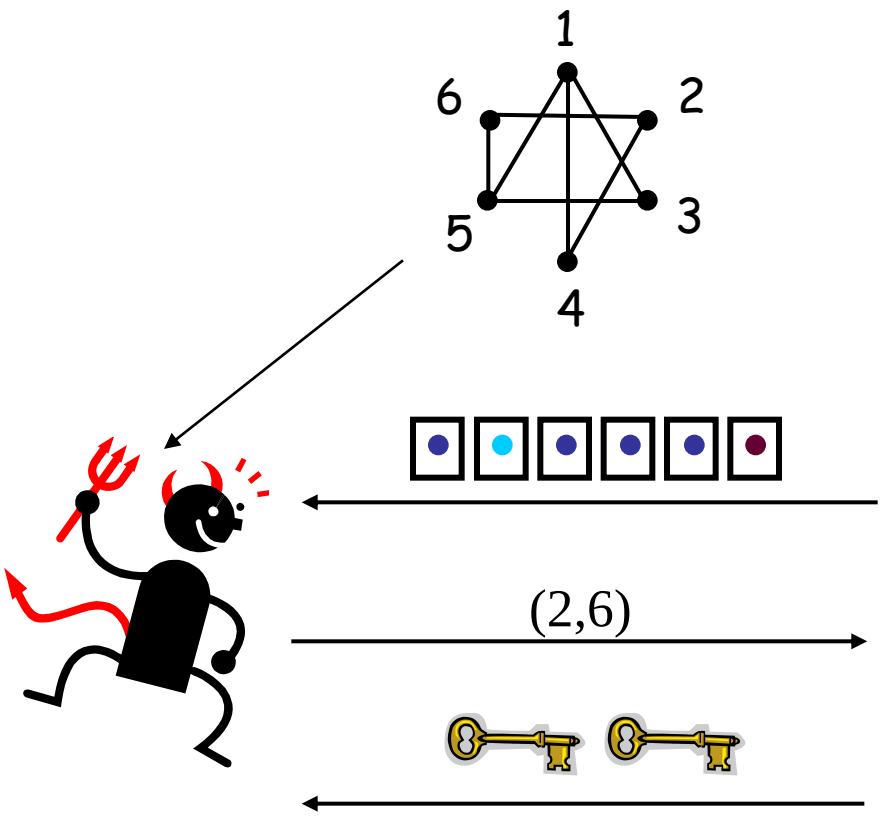


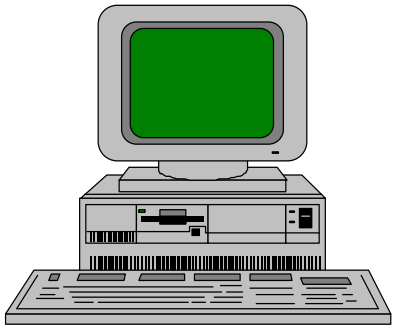



Def: (P,V) is zero knowledge if
∀ poly-time V* ∃ poly-time S s.t. when $x \in L$
S(x) is computationally indistinguishable from $(P,V^*)(x)$



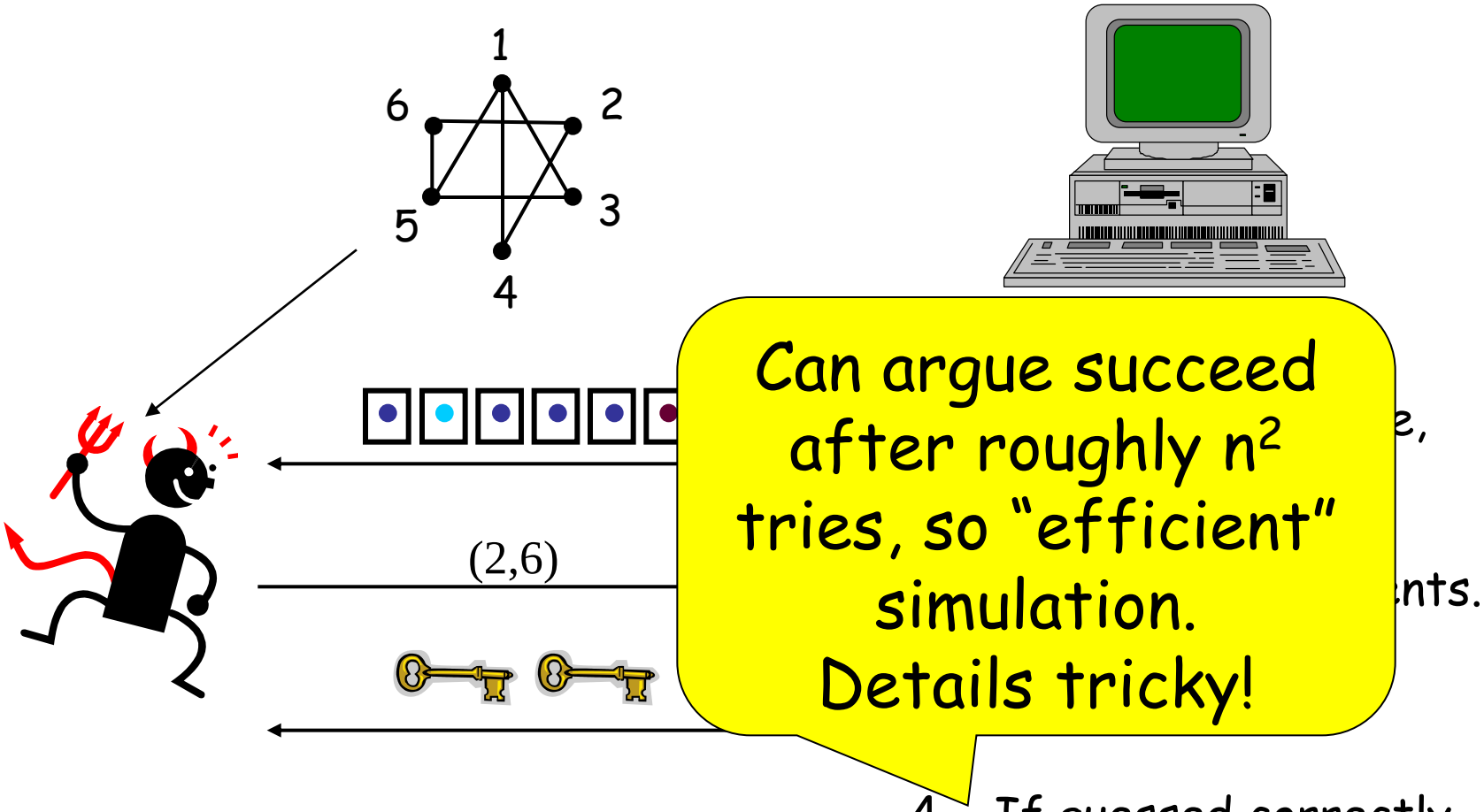
# Simulator for 3-COL Protocol



1. Randomly guess verifier's challenge, e.g. (2,6)
2. Prepare commitments.
3. Run verifier.
4. If guessed correctly, complete simulation. Else try again.

# Simulator for 3-COL Protocol




4. If guessed correctly, complete simulation. Else try again.

# Flavors of Zero-Knowledge Proofs

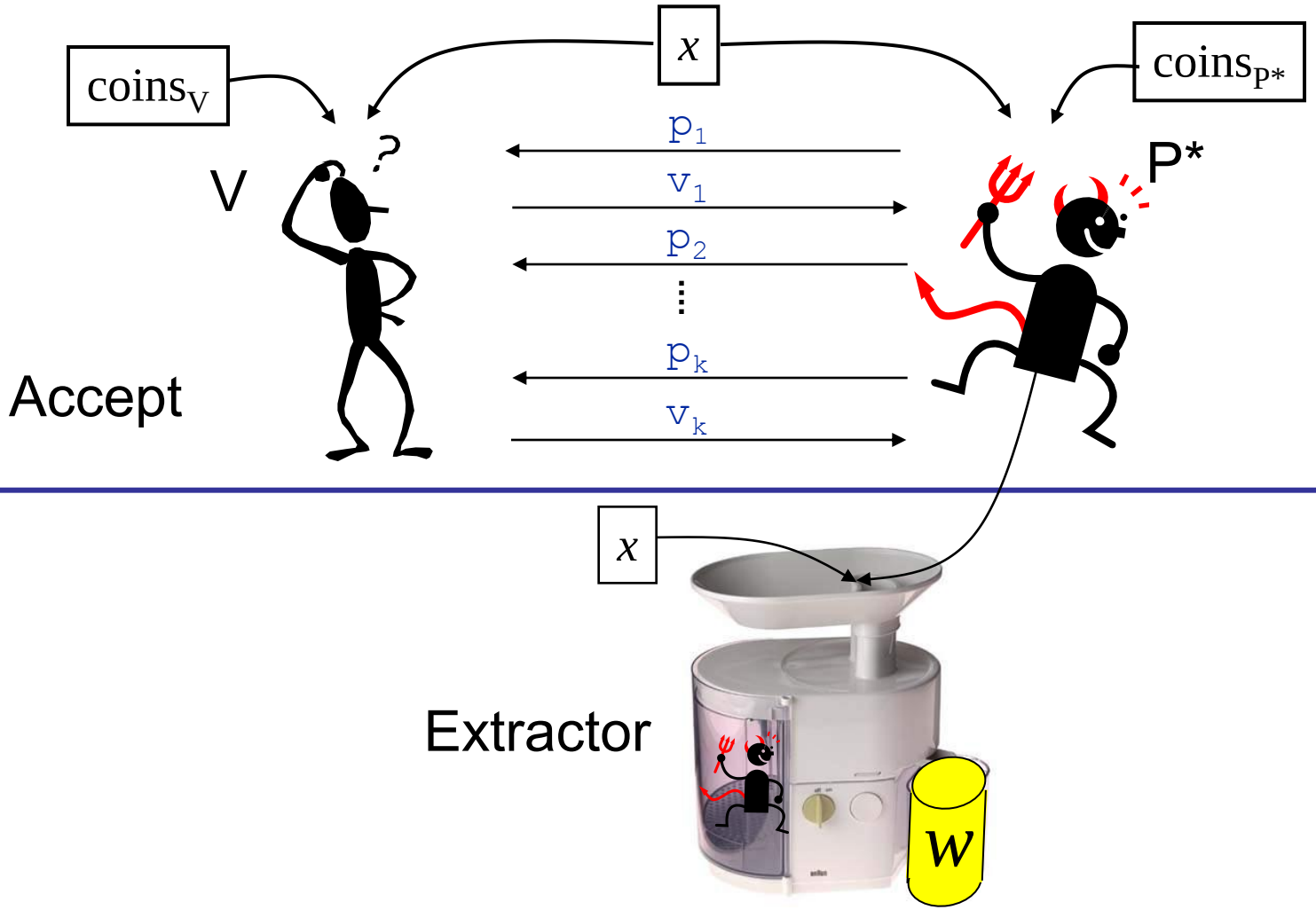
- Quality of ZK/Simulation:
  - Perfect (PZK), Statistical (SZK), Computational (ZK)
- Verifier strategies considered:
  - Public-coin (ala GI) vs. private-coin (ala GNI).
    - Turns out equal (costs at most 2 rounds!) [GS90]
  - Honest-verifier ZK (HVZK) vs. general ZK (ZK)
- Prover Efficiency:
  - Poly-time (for NP, with witness) vs. unbounded

# Flavors of Zero-Knowledge Proofs

- Quality of ZK/Simul
  - Perfect (PZK), Stati
- Verifier strategies
  - Public-coin (ala GI) v
    - Turns out equal (co
  - Honest-verifier ZK (H
- Prover Efficiency:
  - Poly-time (for NP, with witness) vs. unbounded
- Soundness:
  - Proof systems: secure even against unbounded provers
  - Arguments: only sound against poly-time provers
    - Why? Efficiency, can do with polylog communication [Kil92,Mic92]!
  - Proofs/arguments of knowledge
    - P not only proves that the statement is true, but that he knows the witness !

Thm [BP92,V04,NV05]:
Every $\Pi \in ZK \cap NP$ has a zero-knowledge proof where prover can be implemented in poly-time given an NP witness.

Def: (P,V) is Proof of Knowledge (PoK)
if ∃ poly-time extractor E s.t. ∀ poly-time P*,
if P* convinces V with non-negligible probability,
then $E^{P^*}(x)$ outputs a witness w for x



# Are current ZK proofs also PoK?

- NO! Look at GI and 3-COL protocols
  - Easy to cheat with probability $\frac{1}{2}$ and $(1 - 1/|E|)$, respectively !
    - Just guess the verifier challenge and run the simulator
  - These correspond precisely to the soundness of the ZK proof

- More robust definition: (P,V) is a Proof of Knowledge with error p if ∃ poly-time extractor E s.t. ∀ poly-time P*, if P* convinces V with probability non-negligibly greater than p, then $E^{P^*}(x)$ outputs a witness w for x
  - Note: PoK with error p implies soundness error p

- Now, can show GI and 3-COL are PoKs with error $\frac{1}{2}$ and $(1-1/|E|)$, respectively
  - If P* can show a graph isomorphic to both $G_1$ and $G_2$, then easy to recover isomorphism between $G_1$ and $G_2$

# Reducing the Error?

- Make sense for both soundness and PoK
  - For PoK can even try reducing to 0 (impossible for soundness)
- Sequential Repetition: always works, but many rounds 😐
- Parallel Repetition:
  - Works for soundness ☺, does not always work for PoK ☹
  - Works for PoK of specific protocols, including GI and 3-COL ☺
- Problem: does not necessarily preserve zero knowledge !
  - Honest-verifier ZK is preserved with parallel repetition, but…
  - For general ZK cannot guess a long challenge with good prob.
  - In fact, parallelized 3-round 3-COL protocol is unlikely to be ZK

# Reducing the Error?

- Make sense for both soundness and PoK
  - For PoK can even try reducing to 0 (impossible for soundness)
- Sequential Repetition: always works, but many rounds 😐
- Parallel Repetition:
  - Works for soundnes ☹
  - Works for PoK of sp 3-COL ☺
- Problem: does not n wledge !
  - Honest-verifier ZK i on, but…
  - For general ZK cannot gu long challenge with good prob.
  - In fact, parallelized 3-round 3-COL protocol is unlikely to be ZK
- Question: can we have constant-round ZK proof (or PoK) with negligible soundness error? (stay tuned)

Thm [GK90]:
Only "trivial" languages have 3-round "black-box" ZK proofs with negligible soundness error

# Important Issues

- Soundness error
  - Can be reduced by sequential repetitions
  - ZK not preserved under parallel repetition [FS90,GK90]
- Interaction
  - Constant rounds with negligible error? [FS89,GK88]
  - Non-Interactive (one-message) ZK (NIZK)? [BDMP91]
- Composability
  - Design protocols that remain ZK under concurrent, person-in-the-middle, reset, … attacks. [F90,DNS98,DDN90,CGGM00,…]
- Efficiency
  - Practical protocols for specific problems (like e-cash, anonymous credentials)
- Minimizing complexity assumptions
  - $\exists$ commitment schemes $\Leftrightarrow$ $\exists$ one-way functions $\Rightarrow$ P$\neq$NP
- Deniability and Transferability of Proofs

# Important Issues

- Soundness error
  - Can be reduced by sequential repetitions
  - ZK not preserved under parallel repetition [FS90,GK90]

- Interaction
  - Constant rounds with negligible error? [FS89,GK88]
  - Non-Interactive (one-message) ZK (NIZK)? [BDMP91]

- Compo
  - De n-the-mi

- Effici
  - Pr mous credentials)

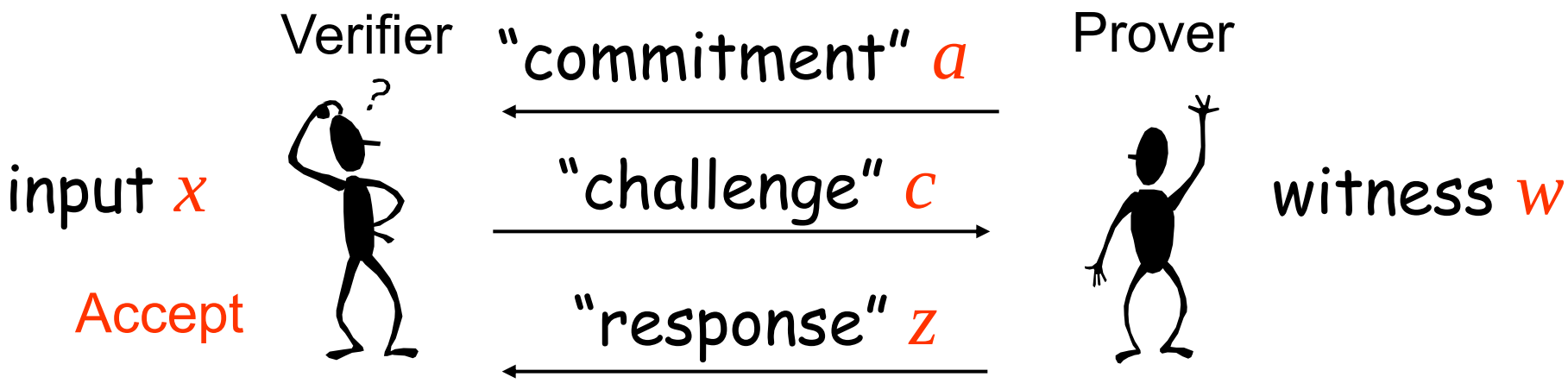
- Minimizing complexity assumptions
  - ∃ commitment schemes ⇔ ∃ one-way functions ⇒ P≠NP

- Deniability and Transferability of Proofs

**Many of these questions can be (partially) answered using Σ–protocols !**

# Σ–Protocols

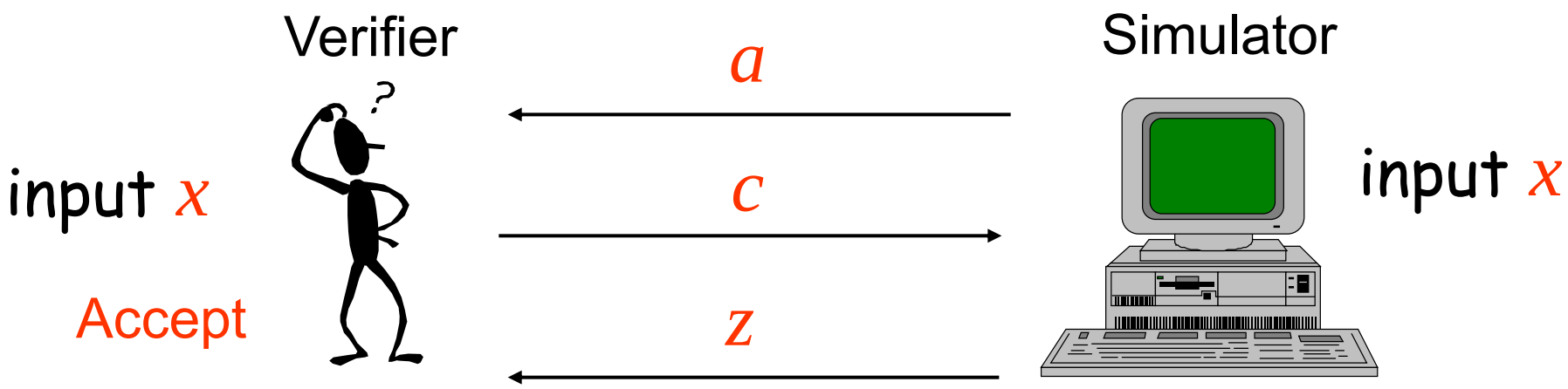
- "Special" 3-round HVZK PoK:




- Special HVZK:
  - Know $c$ in advance $\Rightarrow$ can fake proofs for any $x$, even $x \notin L$

# $\Sigma$–Protocols

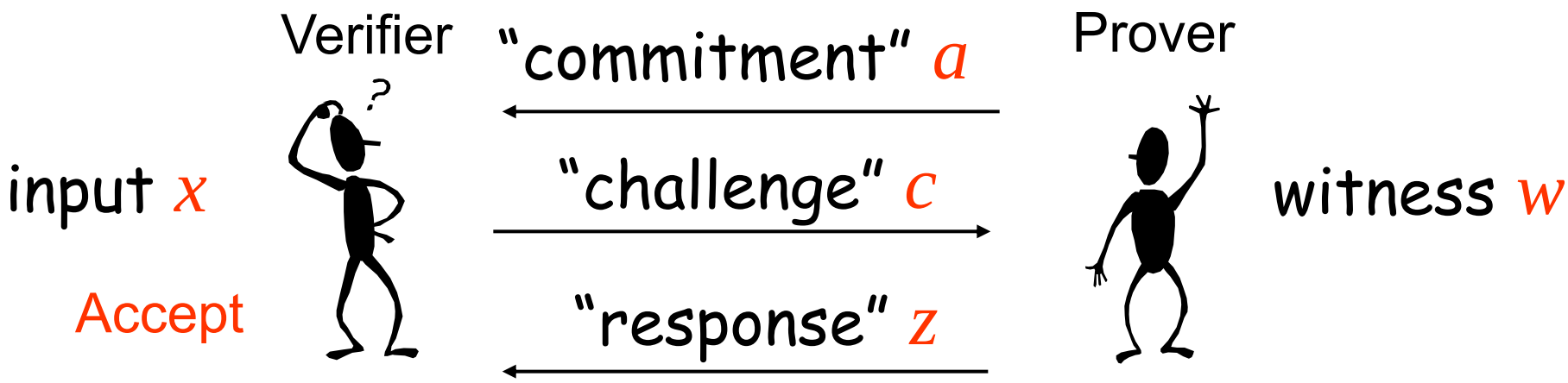
- "Special" 3-round HVZK PoK:




- Special HVZK:
  - Know $c$ in advance $\Rightarrow$ can fake proofs for any $x$, even $x \notin L$
  - Implies HVZK: Sim picks random $c$ and fakes consistent $(a,z)$
  - Not general ZK: what if $c$ depends on $a$ ?

# $\Sigma$–Protocols

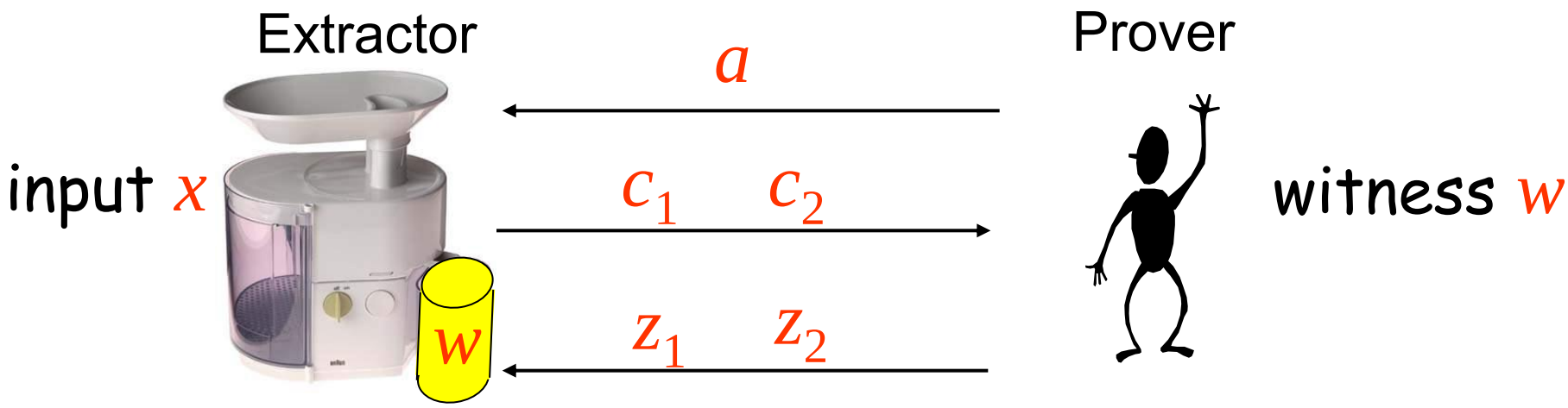
- "Special" 3-round HVZK PoK:




- Special HVZK:
  - Know $c$ in advance $\Rightarrow$ can fake proofs for any $x$, even $x \notin L$
- Special Soundness:
  - Know two distinct conversations with same $a$ $\Rightarrow$ recover witness $w$

# Σ–Protocols

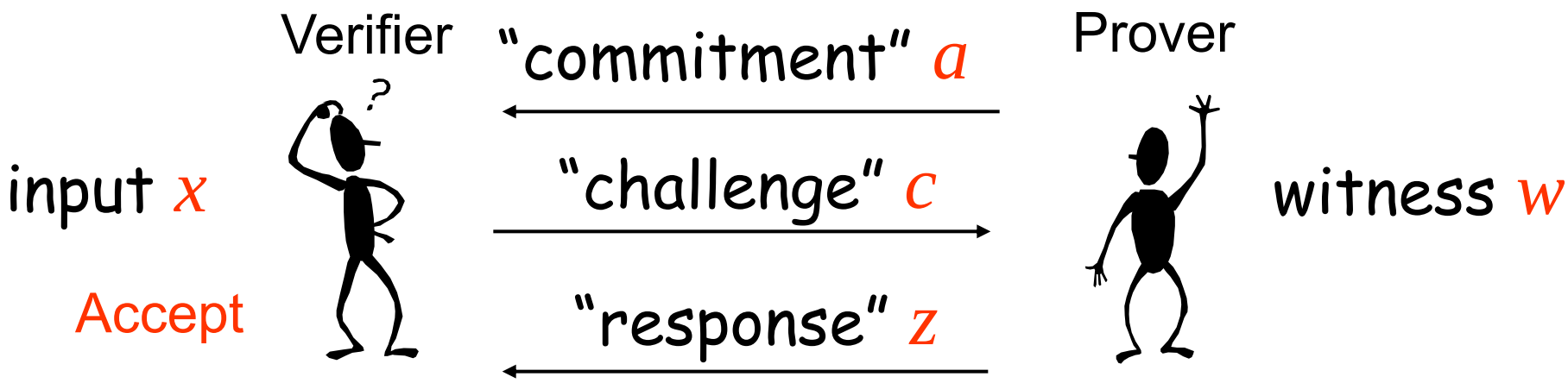
- "Special" 3-round HVZK PoK:




- Special HVZK:
  - Know $c$ in advance ⇒ can fake proofs for any $x$, even $x \notin L$
- Special Soundness:
  - Know two distinct conversations with same $a$ ⇒ recover witness $w$
  - Implies soundness/knowledge error = 1/#challenges when $x \notin L$

# $\Sigma$–Protocols

- "Special" 3-round HVZK PoK:




- Special HVZK:
  - Know $c$ in advance $\Rightarrow$ can fake proofs for any $x$, even $x \notin L$
- Special Soundness:
  - Know two distinct conversations with same $a$ $\Rightarrow$ recover witness $w$

# Example: Knowledge of DL

input
$y = g^x$

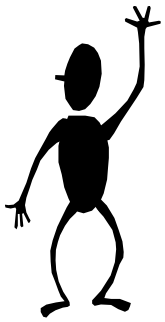
Verifier

$a = g^r$

random $c$

$z = r - cx$

Prover



witness $x$

Accept iff
$a = g^z y^c$

- Special HVZK:
  - Know $c$ in advance $\Rightarrow$ pick random $z$, and let $a = g^z y^c$
- Special soundness:
  - Know accepting $(a, c_1, z_1), (a, c_2, z_2) \Rightarrow a = g^{z_1} y^{c_1} = g^{z_2} y^{c_2}$

$$\Rightarrow x = (z_1 - z_2) / (c_2 - c_1)$$

# Example: Knowledge of DL

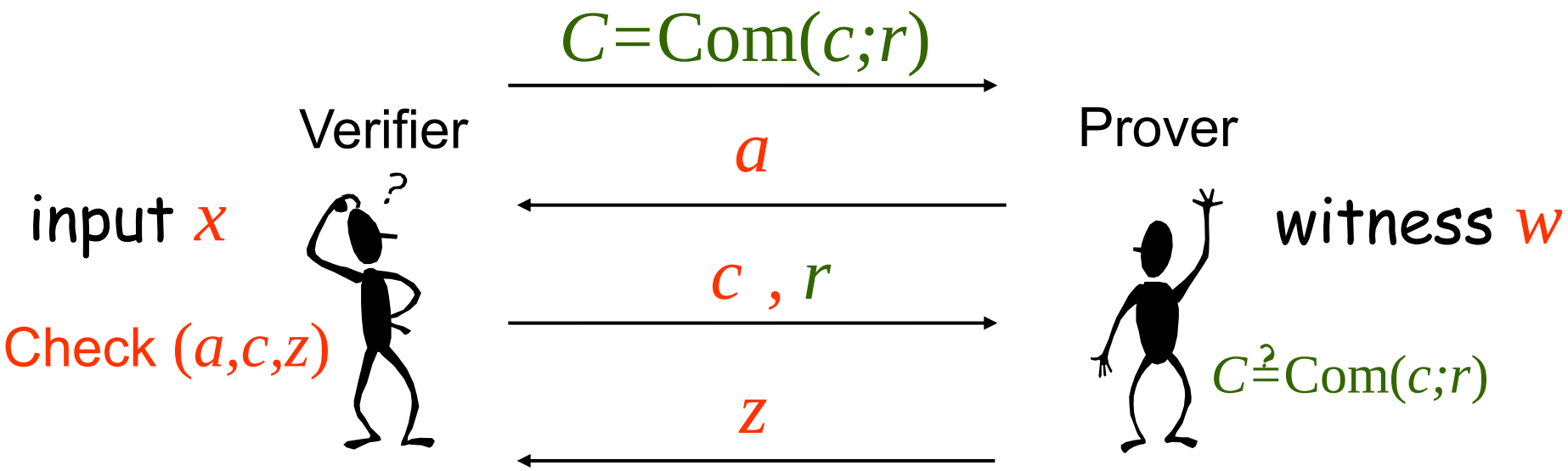
- Notice, corresponds to trivial language, but PoK aspect non-trivial
- Generalizes to proving knowledge of discrete log representation (very useful for commitments)
- With more work, can prove extremely non-trivial relations in the exponent (e-cash, etc. come from here)
- **Thm**: OWFs $\Rightarrow$ $\Sigma$–Protocols for any $L \in NP$ (no special soundness in 3COL protocol, use HamCycle instead [FLS90])

# More on Σ–Protocols

- Most abundant technique for building ZK proofs!
  - GI example was a Σ-protocol (3-COL was not, but "close")
  - Can have Σ-protocols for NP-complete languages (Hamiltonicity)
  - Very useful (and natural) for number-theoretic relations
- Properties of Σ-protocols:
  - Can be repeated in parallel (maintains special HVZK/soundness↓)
  - Σ-protocols for A and B $\Rightarrow$ Σ-protocols for $(A \wedge B)$ and $(A \vee B)$.
    - For $(A \wedge B)$, V uses same challenge c for A and for B
    - For $(A \vee B)$, let P split $c = c_1 + c_2$ and prove A with $c_1$, B – with $c_2$
  - Witness-indistinguishable, very useful for $(A \vee B)$:
    - E.g., "Either I know my Sig(m) or I know your secret key"
- Useful for more advanced flavors of ZK, but also other things (signatures, trapdoor commitments, …)

# Constant-Round ZK [GK88]

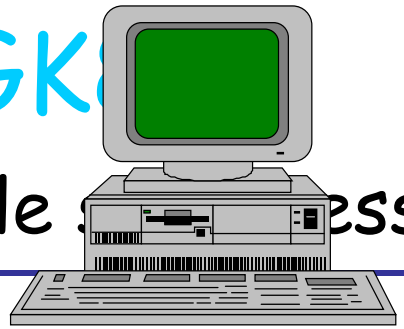
- Start with $\Sigma$-protocols w./negligible soundness

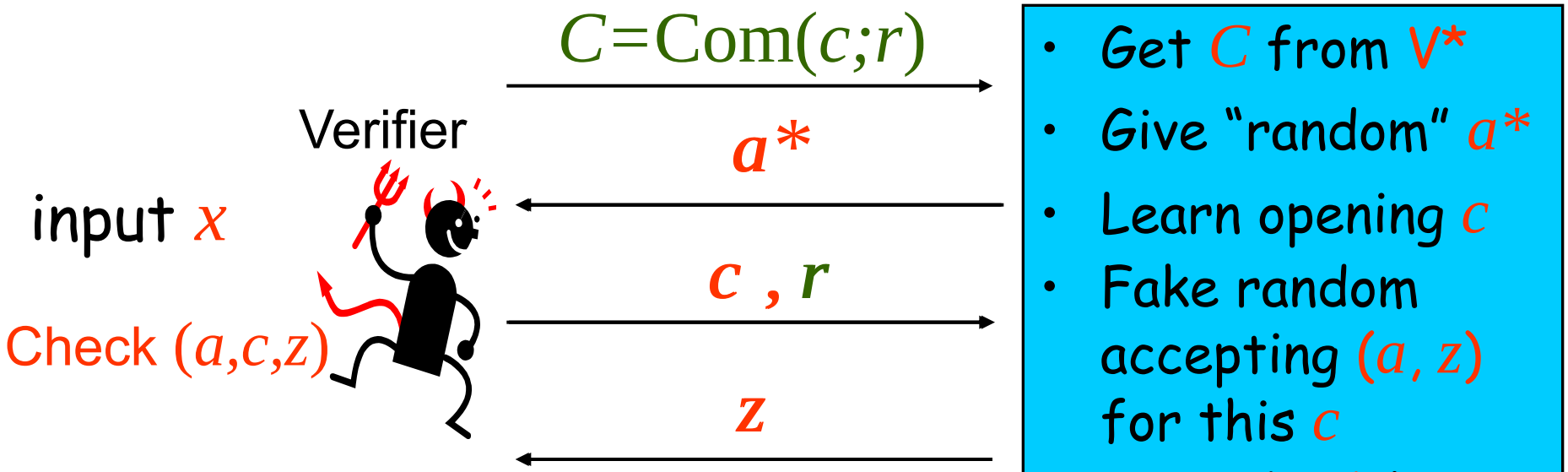



- Add commitment round: let V commit to his challenge $c$
- Then open it after P sent his commitment $a$
- P will also check that the commitment is properly opened
- Soundness: by hiding of $C$, flow $a$ still cannot depend on $c$

# Constant-Round ZK [GK

- Start with Σ-protocols w./negligible ess



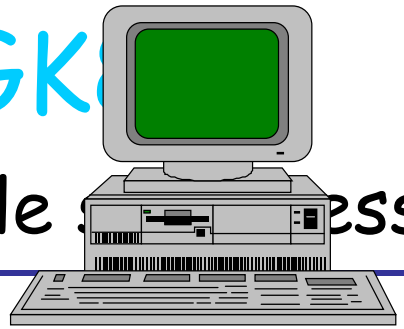


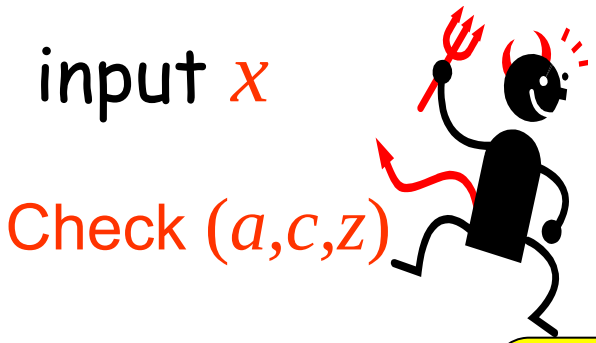

- Add commitment round: let V commit
- Then open it after P sent his commitm
- P will also check that the commitment
- Soundness: by hiding of C, flow a still
- ZK: Simulator can "extract" c from V* and "rewind"

# nd ZK [GK

- w./negligible ess

Destroyed PoK aspect. Can "fix" using a few more rounds.






- Get $C$ from V*
- Give "random" $a^*$
- Learn opening $c$
- Fake random accepting $(a, z)$ for this $c$
- Rewind V* !
- Use $(a, z)$ with V* on the same $C$
- "OK" as V* must still open $C$ to $c$

Formal analysis tricky: what if V* refuses the second time?

- Add commitm
- Then open it
- P will also check that the commitmen
- Soundness: by hiding of $C$, flow $a$ still
- ZK: Simulator can "extract" $c$ from V* and "rewind"

# Concurrency

- What if several ZK proofs are going on simultaneously?
- Current ZK definition not good enough if malicious verifiers coordinate their actions
  - similar problem for PoK with malicious provers
- Technical problem: rewinding
  - Can schedule executions requiring exponentially many rewinds (each new rewind forces to redo all the previous ones)
- Leads to concurrent ZK (concurrent PoK)
  - Much harder than (standalone) ZK/PoK
  - Need super-logarithmic number of rounds (can match, in theory)
  - Still uses "clever" rewinding, not good for on-line-deniability
- Can we have straight-line simulation/extraction?
  - Gives on-line deniability (+concurrency, can simulate as you go) ☺
  - Much harder, seemingly impossible ☹

# Attempt 1: Make it Non-Interactive!

- Fiat-Shamir: take $\Sigma$-protocol and make $c = Hash(a)$
  - Extends to signature schemes (GQ, Schnorr) if $c = Hash(a,m)$ !
- Why is this good?
  - Challenge $c$ still chosen "after" $a$ in "unpredictable" manner
  - No longer need V during proof (only to verify) !
  - Means get general ZK, not just HVZK (no malicious V) !
  - In fact, event concurrent ZK (no issue of scheduling) !
- Why is this bad?
  - Can only [GT03] analyze in the random oracle model
  - Alternatively, if view ***Hash*** as a "trusted setup"
  - Lose deniability, proof is now transferable !
- Can we replace random oracle with a "cheaper" setup?
  - Yes, Common Reference String (CRS) model

# Non-Interactive ZK with a CRS

- [BFM88]: assume trusted party publishes a CRS
  - Both Prover and Verifier trust it was properly generated
  - Mild/realistic assumption…
- Where is the "cheating"/"meat"?
  - When simulating, Sim can choose his own CRS' ($\approx$ CRS)
  - In particular, can plant a trapdoor not known to the real Prover, which allows it to cheat
- Example: instead of proving $x$, prove $x'$ = "either $x$ or CRS is pseudorandom"
  - Now, Sim has a witness to $x'$, without having a witness to $x$
  - However, Prover must still prove $x$ (real CRS is actually random)
- Deniability lost since Sim cannot work with "real" CRS
- Also quite inefficient in practice… ☹ (getting better)

# Regaining Efficiency with Interaction

- [Dam01]: can we have fast interactive concurrent ZK in the CRS model?
  - Yes, add a tweak to any $\Sigma$-protocol !
- Let CRS = (public) key PK to a trapdoor commitment scheme
  - Prover cannot can break binding in the real model
  - Simulator knows trapdoor TK that can open commitments arbitrarily
- Example: Pedersen, $\mathrm{Com}(b; K) = g^K h^b \bmod p$
  - If know $m = \mathrm{Dlog}_g(h)$, can open $g^K$ to 0 ($K$) or 1 ($K - m$)
- Aside: how to build other trapdoor commitments?
  - use $\Sigma$-protocols again (Pedersen is special case!)

# Concurrent ZK in the CRS Model

- Instead of sending $a$, send $C = \mathrm{Com}_{\mathrm{PK}}(a)$
- In last flow, open $C$ to $a$ together with sending $z$
- In real world, still sound as before, since any Prover is bound to some $a$ by $C$ (in fact, PoK)
- But Simulator can simulate V* straight-line:
  - Commit to garbage in first flow
  - Learn $c$ from malicious verifier V*
  - Prepare fake $(a, z)$ and open $C$ to $a$ (w. trapdoor $\mathrm{TK}$) !
- More work for concurrent extraction
- Still not deniable in real world…

# Further Reading

- O. Goldreich. *Foundations of Cryptography — Volume I (Basic Tools).* Cambridge University Press, 2001.

- O. Goldreich. "Zero knowledge twenty years after their invention." Tutorial at FOCS `02.