Lectures 2+3: Provable Security

Contents

1 Motivation

The design of cryptographic algorithms presents certain unique challenges. Over time, we have learned that our intuition about what is secure and what is not is very poor. The history of cryptography is filled with examples of cryptosystems that were thought to be highly-secure but turned out to be completely broken. In addition, unlike other areas of computer science, we have no way of testing whether a cryptographic algorithms is secure or not. This lack of feedback on an already unintuitive design process can lead to serious problems. Another difficulty is that cryptographic algorithms are fragile in the sense that even slight and seemingly inconsequential changes to a secure primitive can completely break it. Finally, the skillsets needed to *design* a cryptosystem are not necessarily the same as the ones needed to *break* a cryptosystem. Because of this, cryptographic designers cannot rely solely on their own expertise to reason about the security of their algorithms.

Provable security. To address these problems, cryptographers have developed a methodology with which to analyze the security of cryptographic algorithms. It is usually referred to as the *provable security* paradigm. The term provable security, however, can be misleading to non-cryptographers so an alternative name you might hear is the *reductionist security paradigm*. The issue with the name *provable* security is that one might interpret it to mean that schemes that have been analyzed with this approach are secure. While this is sometimes the case, what the provable security paradigm usually guarantees is that *a scheme is secure*

relative to a specific security definition against a given adversarial model and under a particular assumption. The hope, of course, is that the security definition is appropriate for the intended application, that the adversarial model captures real-world adversaries and that the assumption is reasonable. But provable security gives us more than a security guarantee. It also gives cryptographers a way to *debug* their work. When one tries to prove the security of a scheme and fails, it is a hint that there is a subtle security issue with the scheme and this feedback is often crucial in improving the algorithm. This is particularly important since we don't have any way of testing the security of cryptographic schemes. In the rest of this Section, we will go over concrete examples of some of the subtleties that can come up when designing cryptographic algorithms.

Textbook RSA. The textbook version of RSA works as follows. To encrypt a message m from the message space $\mathbb{Z}_N^*$, where $N = pq$ with p and q prime and $|N| = 2048$, one computes the ciphertext $c = m^e \mod N$, where $1 < e < \phi(N)$ and $\phi(N) = (p-1)(q-1)$. The secret key is $\text{sk} = (N, d)$, where d is such that $ed = 1 \mod \phi(N)$, and the public key is $\text{pk} = (N, e)$. To decrypt, one computes $m := c^d \mod N$. Under the assumption that it is hard to compute eth roots modulo N, the RSA encryption scheme seems secure. Indeed, one can show that under this assumption it is not possible to recover m from c.

It turns out, however, that given an RSA ciphertext c one can learn *something* about m. Not necessarily everything about m but something nonetheless. Specifically, given a ciphertext c, one can compute the Jacobi symbol of the message over N. The Jacobi symbol is a number-theoretic property related to whether a number is a square modulo N or not (for more on Jacobi symbols see the Wikipedia page). Another problem with vanilla RSA is that if the message space is small, say $|N| = 10$, then one can do a brute force attack as follows. Given a ciphertext c and the public key $\text{pk} = (N, e)$, just compute $c' := m^e \mod N$ for every message $m \in \mathbb{Z}_N^*$ and check to see if $c' = c$. These issues clearly demonstrate that vanilla RSA is not really secure; at least not in a satisfactory way. It turns that there are many more issues with vanilla RSA.

The Advanced Encryption Standard (AES). AES is a block cipher that was designed by Daeman and Rijmen. A block cipher takes as input a B-bit block m and a secret key K and returns a ciphertext ct. AES encrypts 128-bit blocks and has three possible key sizes: 128, 192 and 256 bits. It is widely believed to be secure in a sense that we will make formal later. But what if we want to encrypt more than 128 bits? In this case, we use a *mode of operation*. A mode of operation turns a block cipher, which encrypts fixed-sized messages, into an *encryption scheme*, which encrypts variable-sized messages. The most natural mode of operation is called *encrypted codebook* (ECB) mode. It works as follows: parse the message m into several B-bit blocks and apply the cipher to each block. The intuition is that if the cipher is secure then the entire ciphertext will be secure. Unfortunately, this intuition is incorrect because if two blocks in m are the same, their encryptions will also be the same. In other words, given an ECB encryption we can immediately tell whether the plaintext has any repeated blocks or not. This illustrates a basic and important problem in cryptography: *it is*

very easy to take a secure building block (e.g., AES) and use it in a way that is completely insecure. We will see this a lot in the encrypted search literature so it is important to be aware of this.

Overview. Hopefully the previous examples were enough to convince you that simply proclaiming a cryptosystem secure because it "feels" secure or because it is based on secure building blocks is not a good idea. The correct way to analyze the security of a cryptosystem is to, whenever possible, use the provable security paradigm which includes the following steps:

1. Define the *syntax* of the cryptographic object being studied;
2. Formulate an appropriate definition of *correctness* for this object;
3. Formulate an appropriate definition of *security* for this object;
4. *Prove* that a specific instantiation meets this definition.

We'll now go over all these steps in detail using RSA as a concrete example.

2 Syntax

The syntax of a cryptosystem can be thought of as its API. It specifies the algorithms in the cryptosystem and their inputs and outputs. When discussing syntax, we are focusing on the object and not on a specific instantiation of that object. For example, RSA is an instantiation of a public-key encryption scheme and AES (with a mode of operation) is an instantiation of a secret-key encryption scheme. At this level, we don't care about the details of RSA or AES, only that they are public-key and secret-key encryption schemes, respectively. We now give an example of a syntax definition.

Definition 2.1 (Secret-key encryption)**.** *A secret-key encryption scheme* $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *consists of three polynomial-time algorithms that work as follows:*

- $K \leftarrow \mathsf{Gen}(1^k)$*: is a probabilistic key generation algorithm that takes as input a security parameter* 1^k *and outputs a secret key* K*.*
- $\mathrm{ct} \leftarrow \mathsf{Enc}(K, m)$*: is a probabilistic encryption algorithm that takes as input a secret key* K *and a message* m *and outputs a ciphertext* ct*. We sometimes write this as* $\mathrm{ct} \leftarrow \mathsf{Enc}_K(m)$*.*
- $m := \mathsf{Dec}(K, \mathrm{ct})$*: is a deterministic decryption algorithm that takes as input a secret key* K *and a ciphertext* ct *and outputs a message* m*. We sometimes write this as* $m := \mathsf{Dec}_K(\mathrm{ct})$*.*

Notice that Definition 2.1 is *asymptotic*; for example, we require that the algorithms run in polynomial-time. Modeling cryptosystems asymptotically has several consequences and it is important to be aware of the tradeoffs it imposes. On one hand, it simplifies analysis and proofs; on the other, we lose some connection to the real world.

The security parameter. In cryptography, we design primitives and protocols in such a way that we can tune how much security they provide. This is done using a *security parameter*, typically denoted k. Intuitively, the larger k is, the more secure the scheme is (of course we haven't defined what we mean by secure yet) and the slower the scheme is. So, for example, RSA with $k = 2048$ is more secure but slower than RSA with $k = 1024$; and AES with $k = 256$ is more secure but slower than AES with $k = 128$. What exactly counts as the security parameter changes from scheme to scheme. For example, in RSA the security parameter is the bit-length of the modulus N whereas in AES it is the bit-length of the key.

In the asymptotic framework, one assumes the security parameter is given to each algorithm (including the adversary) and we measure efficiency and security as functions of k. Since every algorithm takes k as input, we mostly don't bother to writing it explicitly in syntax definitions; except for algorithms that setup a scheme like key generation algorithms.

One question you may have is why we give the Gen algorithm 1^k (which denotes k is unary) and not k? The reason is that the asymptotic running time of an algorithm is normally defined as a function of its input length. So, under this convention, if we passed k to the algorithm it would have to run in time polynomial in $\log(k)$ which is not what we want. So we pass k in unary because the bit-length of 1^k is k.

Probabilistic polynomial time. In this framework we say that a scheme is efficient if it runs in probabilistic polynomial-time (PPT). In other words, if it is allowed to use randomness and if it has running time $k^{O(1)}$. We often write $\mathsf{poly}(k)$ to mean $k^{O(1)}$. In these notes we denote the output of a randomized algorithm by $\leftarrow$ as in $\mathrm{ct} \leftarrow \mathsf{Enc}_K(m)$ and the output of a deterministic algorithm by $:=$ as in $m := \mathsf{Dec}_K(\mathrm{ct})$. When we need to make the randomness of an algorithm explicit we write $\mathrm{ct} \leftarrow \mathsf{Enc}_K(m; r)$, where r denotes the random coins used by the algorithm.

Negligible functions. In asymptotic cryptography, we usually say that a scheme is secure if the adversary's probability of breaking it is *negligible* in the security parameter k. More precisely, if the probability that any PPT adversary breaks the scheme is a negligible function in k. A function is negligible if it is in $1/k^{\omega(1)}$.

Definition 2.2 (Negligible function)**.** *A function f is negligible if for every constant $c \in \mathbb{N}$, there exists some $k_c \in \mathbb{N}$, such that for all $k > k_c$, $f(k) < 1/k^c$.*

Another way of expressing this is that the function gets smaller faster than 1 over any polynomial in k. For example, an event that occurs with probability $1/2^k$ occurs with negligible probability but one that occurs with probability $1/\mathsf{poly}(k)$ does not.

It is important to understand why we define negligible in this way. The reason is so that we can guarantee that a polynomial-time adversary cannot amplify its success probability. For example, suppose a PPT adversary can break a scheme with probability $1/\mathsf{poly}(k)$. Then it could just attack the scheme $\mathsf{poly}(k)$-many times and hope that it works at least once. What is the probability that the adversary successfully breaks the scheme? Let n be the number of attempts, break be the event that the adversary is successful, break_i be the event that it is successful on the ith attempt and let p be the probability of success. Assuming the attempts are independent, we have

$$\Pr[\,\mathsf{break}\,] = \Pr\left[\bigvee_{i=1}^{n} \mathsf{break}_i\right] = 1 - \Pr\left[\bigwedge_{i=1}^{n} \overline{\mathsf{break}_i}\right] = 1 - (1-p)^n \geq 1 - \frac{1}{e^{np}},$$

where the last inequality follows from $(1-p)^n \leq e^{-np}$. If $p = 1/\mathsf{poly}(k)$, then we can write it as $1/k^{c_1}$, for some constant $c_1 \in \mathbb{N}$. And since the adversary is PPT, we have $n = \mathsf{poly}(k) = k^{c_2}$ for some constant $c_2 \in \mathbb{N}$. We therefore have $np = k^{c_2 - c_1}$. Note that if the adversary sets $c_2 = c_1 + 1$ (i.e., it chooses to make $n = k^{c_2} = k^{c_1+1}$ attempts) its probability of success will be $1 - 1/e^k$ which goes to 1 exponentially-fast in k. On the other hand, we also have the following upper bound,

$$\Pr[\,\mathsf{break}\,] = \Pr\left[\bigvee_{i=1}^{n} \mathsf{break}_i\right] \leq \sum_{i=1}^{n} \Pr[\,\mathsf{break}_i\,] \leq np,$$

where the first inequality follows from the union bound. So if $p = \mathsf{negl}(k)$, this strategy will work with probability at most $\mathsf{poly}(k) \cdot \mathsf{negl}(k)$ which is itself negligible (and decreases very quickly in k).

3 Correctness

The second step of the provable security paradigm is to formulate a notion of correctness. This describes precisely how the object should function so that it is usable. This has nothing to do with security.[1] In the case of encryption (public-key or secret-key) correctness means that if we use the appropriate keys we should always be able to decrypt what we encrypt. This is formalized as follows.

Definition 3.1 (Correctness)**.** *A secret-key encryption scheme* $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is correct if for all* $k \in \mathbb{N}$*, for all keys* K *output by* $\mathsf{Gen}(1^k)$*, for all messages* $m \in \mathbb{M}_k$*, for all ciphertexts* ct *output by* $\mathsf{Enc}_K(m)$*,* $\mathsf{Dec}_K(\mathrm{ct}) = m$*.*

The definition is formulated in such a way that no matter how we choose the security parameter and the message and no matter what randomness Gen and Enc use, we will always correctly decrypt a validly encrypted message. Often, the correctness of a concrete construction is obvious so we don't bother proving it. Also, note that while here we require

[1] In some cases it does but for now let's just assume it doesn't.

decryption to always work, when we are considering more complex objects (e.g., encrypted search algorithms) we will sometimes allow operations to be correct with probability only close to 1.

4 Security Definitions

The third step of the provable security paradigm is to give a security definition. Syntax and correctness are relatively straightforward once you get the hang of it but security definitions are more complicated and much more subtle. When formulating a security definition you have to proceed in two phases: a conceptual phase and a formal phase.

In the conceptual phase, you need to formulate an intuition of how a secure object should behave when interacting with the adversary. Note that even this intuitive phase can be tricky as illustrated by the RSA and AES examples from Section 1. Once you have a good understanding of how the object should behave in adversarial environments, you need to come up with a way of formalizing that intuition. In cryptography, we typically use one of two approaches to formalize security intuitions. We'll discuss both.

Game-based definitions. In a game-based definition, security is formalized as a game against an adversary. In this game, the adversary interacts with the cryptosystem in a specific way. If the adversary wins the game, the scheme is deemed insecure and if it loses the game, the scheme is deemed secure. Here is an example for secret-key encryption.

Definition 4.1 (Indistinguishability)**.** *Let* $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a secret-key encryption scheme and consider the following randomized experiment against a* stateful PPT *adversary* $\mathcal{A}$*:*

$\mathbf{CPA}_{\mathcal{A}}(k)$*:*

1. *a key* $K \leftarrow \mathsf{Gen}(1^k)$ *is generated;*
2. $\mathcal{A}$ *chooses poly-many messages and receives their encryptions;*
3. $\mathcal{A}$ *chooses two equal-length challenge messages* m_0 *and* m_1*;*
4. *a bit* $b \xleftarrow{\$} \{0,1\}$ *is sampled and a challenge ciphertext* $\mathrm{ct} \leftarrow \mathsf{Enc}_K(m_b)$ *is computed;*
5. $\mathcal{A}$ *is given* ct*, chooses poly-many messages and receives their encryptions;*
6. $\mathcal{A}$ *outputs a guess* b'*;*
7. *if* $b' = b$ *the experiment returns* 1 *else it returns* 0*.*

We say that SKE *is indistinguishable against chosen-plaintext attacks (IND-CPA) if for all* PPT *adversaries* $\mathcal{A}$*,*

$$\Pr[\,\mathbf{CPA}_{\mathcal{A}}(k) = 1\,] \leq \frac{1}{2} + \mathsf{negl}(k).$$

Let's parse this definition. The first thing to notice is that $\mathbf{CPA}_{\mathcal{A}}(k)$ outputs 1 if and only if $\mathcal{A}$ guesses b correctly so the probability that $\mathbf{CPA}_{\mathcal{A}}(k)$ outputs 1 is exactly the probability that $\mathcal{A}$ guesses the bit. Notice, however, that $\mathcal{A}$ can guess correctly with probability $1/2$ even without using the ciphertext by either: (1) outputting a random bit; (2) always outputting 1; or (3) always outputting 0. So what we are really interested in is how much better than $1/2$ the adversary can do. If the bound in the Definition is met, then $\mathcal{A}$ guesses correctly with probability at most $1/2 + \mathsf{negl}(k)$. And since we can safely ignore negligible probabilities we have that, for practical purposes, the adversary cannot guess the bit correctly better than $1/2$.

In other words, given an encryption ct of either m_0 or m_1, $\mathcal{A}$ cannot distinguish between the case that $\text{ct} \leftarrow \mathsf{Enc}_K(m_0)$ and that $\text{ct} \leftarrow \mathsf{Enc}_K(m_1)$. This implies, by definition, that ct hides any "distinguishing" information about the messages or alternatively that it does not reveal anything *unique* about m_0 or m_1. [2] This makes sense, because if it did, then $\mathcal{A}$ could immediately distinguish between the two cases. But since this holds over all message pairs m_0 and m_1 (since they are chosen by the adversaries and we quantify over all efficient adversaries) then the ciphertext must hide all information about the messages.

Another important aspect of the experiment is that in Steps 2 and 5 we allow $\mathcal{A}$ to receive encryptions of any messages that it wants. This captures the possibility that $\mathcal{A}$ could improve its guessing probability by using a history of previously encrypted messages. In many real-world scenarios, attackers can trick their targets into encrypting messages so it is very important to capture this in our definitions.

Simulation-based definitions. An alternative way of defining security for an encryption scheme is *semantic security*. Semantic security is an example of a *simulation*-based definition. Often, game-based definitions are easier to use when you are proving the security of a cryptosystem based on some computational assumption. Simulation-based definitions, however, are often easier to use when you are proving the security of a cryptosystem based on another primitive or protocol. In some cases, the simulation-based variant of a security notion is equivalent to the game-based variant but in other cases it is not.

Definition 4.2 (Semantic security)**.** *A secret-key encryption scheme* $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is semantically-secure if for all* PPT *adversaries* $\mathcal{A}$*, there exists a* PPT *simulator* $\mathcal{S}$ *such that for all distribution ensembles* $\mathcal{M} \stackrel{def}{=} \{\mathcal{M}_\}_{k\in\mathbb{N}}$ *over* $\mathbb{M} \stackrel{def}{=} \{\mathbb{M}_k\}_{k\in\mathbb{N}}$*, for all auxiliary information* $z \in \{0,1\}^*$ *and for all polynomially-computable functions* $f : \mathbb{M}_k \to \{0,1\}^*$*,*

$$\Big|\Pr\Big[\,\mathcal{A}\Big(\mathsf{Enc}_K(m), z\Big) = f(m)\,\Big] - \Pr\left[\,\mathcal{S}(|\mathsf{Enc}_K(m)|, z) = f(m)\,\right]\Big| \leq \mathsf{negl}(k),$$

where $K \leftarrow \mathsf{Gen}(1^k)$ *and* $m \leftarrow \mathcal{M}$*. The probabilities here are over the randomness of* $\mathsf{Gen}(1^k)$*, the choice of* m *and the possible randomness of* $\mathcal{A}$*,* $\mathcal{S}$ *and* Enc*.*

The IND-CPA definition captured security by guaranteeing that no adversary can win in a specific game. The semantic security definition, on the other hand, captures security by

[2] By "unique" we mean information about m_0 that does not hold about m_1 and information about m_1 that does not hold about m_0. A concrete example would be some bit that is set to 1 in m_0 and to 0 in m_1.

guaranteeing the existence of a simulator $\mathcal{S}$ that can do anything the adversary can *without seeing the ciphertext*. Again, think about what this implies. It implies that the ciphertext is useless to the adversary and, therefore, cannot reveal any useful information about the message.

Formalizing this intuition is not straightforward and requires the introduction of several new concepts. This includes $\mathcal{M} \stackrel{def}{=} \{\mathcal{M}_k\}_{k\in\mathbb{N}}$ which is an *ensemble* of distributions over the message space $\mathbb{M} \stackrel{def}{=} \{\mathbb{M}_k\}_{k\in\mathbb{N}}$; that is, a collection of distributions $\mathcal{M}_k$ over $\mathbb{M}_k$ for each value of the security parameter k. z is a bit string that captures any a-priori information the adversary could have about the message (e.g., the parity of the message or the language the message is written in) and f describes some partial information about m that the adversary wants to learn. With this in mind, semantic security guarantees that:

> *For all security parameters we choose, for all possible efficient adversaries, no matter which distributions our messages come from, no matter what a-priori knowledge the adversary has about our message, and no matter what information the adversary wants to learn about our message, with very high probability the encryption of the message will be as useful to the adversary as its length.*

This is a strong guarantee (though not the strongest possible) that basically says we can use the encryption scheme safely in most (but not all) situations. Note, however, that if the length of a message happens to be useful to the adversary in learning something about it then this guarantee is not enough. [3]

A note on equivalence. It turns out that for standard encryption schemes, semantic security is equivalent to IND-CPA so it is common to just speak about the notion of CPA-security without distinguishing which formulation one has in mind.

A note on probabilistic encryption. The definition of CPA-security has an important consequence on the design of encryption schemes. In particular, it requires that the encryption algorithms be either randomized or stateful. [4] To see why, suppose we had a scheme $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}^\star, \mathsf{Dec})$ such that $\mathsf{Enc}^\star$ was deterministic and stateless and consider the adversary $\mathcal{A}^\star$ that works as follows in the $\mathbf{CPA}_{\mathcal{A}}(k)$ experiment. It outputs two messages m_0 and m_1 such that $m_0 \neq m_1$ and that $|m_0| = |m_1|$. In Step 5, it asks for an encryption of m_0 and receives ct_0. It then outputs 0 if $\mathrm{ct} = \mathrm{ct}_0$ and 1 if $\mathrm{ct} \neq \mathrm{ct}_0$. This adversary will succeed with probability 1.

5 Important Cryptographic Primitives

The last step of the provable security paradigm is to prove that a given instantiation meets the security definition. We will give an example of such a proof in Section 6 but first we introduce

[3] Typically we don't worry too much about this because messages can always be padded to make them a certain fixed length, but we will see cases where revealing the length of messages can lead to problems.

[4] In the case of public-key encryption schemes it must be randomized.

a few cryptographic primitives that we make use of in our example. These primitives are extremely useful building blocks throughout cryptography and will be used all throughout the course.

Computational indistinguishability. Computational indistinguishability is a fundamental concept in cryptography. Intuitively, it guarantees that two objects cannot be distinguished by any efficient algorithm. Many security definitions in cryptography are formulated using this notion so it is important to understand it. To formalize it, we need the notion of distribution ensembles which are simply collection of probability distributions $\chi = \{\chi_k\}_{k\in\mathbb{N}}$, one for each value of the security parameter.

Definition 5.1. *Two distribution ensembles χ_1 and χ_2 are computationally-indistinguishable if for all* PPT *adversaries $\mathcal{A}$ that outputs a bit,*

$$|\Pr[\,\mathcal{A}(\chi_1) = 1\,] - \Pr[\,\mathcal{A}(\chi_2) = 1\,]| \leq \mathsf{negl}(k).$$

The definition guarantees that no efficient adversary can distinguish between being given a sample from χ_1 or χ_2. This is indeed the case because it outputs 1 (and therefore 0) with roughly the same probability whether it receives a sample from χ_1 or χ_2.

Pseudo-random functions. A pseudo-random function (PRF) is a function that is computationally-indistinguishable from a random function. A random function is a function that is sampled uniformly at random from a finite function space. To make things more concrete, suppose we are interested in a random function from $\{0,1\}^\ell$ to $\{0,1\}^\ell$. One way to think about a random function is as an oracle that: (1) outputs a uniformly random value $y \xleftarrow{\$} \{0,1\}^\ell$ when queried on an input x for the first time; but (2) outputs y every time it is queried on x again. An alternative way to think about a random function is using its "truth" table which consists of a row for every element of its domain that holds the image of the corresponding element. The truth table of a random function is a table where each element is chosen uniformly at random in the co-domain of the function. Random functions are extremely useful in cryptography. We can use them to encrypt, to sign messages, and to generate random numbers and keys. Unfortunately, random functions aren't practical because we can't generate and store them efficiently (i.e., in polynomial time). Consider the table representation of a random function from $\{0,1\}^\ell$ to $\{0,1\}^\ell$. There are 2^ℓ rows and, for each row, 2^ℓ possible values so there are $(2^\ell)^{2^\ell} = 2^{\ell\cdot 2^\ell}$ possible functions. This means that to store one of these functions we need

$$\log 2^{\ell\cdot 2^\ell} = \ell \cdot 2^\ell$$

bits of storage. If $\ell = \Omega(k)$ then we need an exponential number of bits in the security parameter.

Fortunately, we can get around this problem by using *pseudo*-random functions which are efficiently storable and computable functions that are computationally-indistinguishable from random functions.

Definition 5.2 (Pseudo-random functions)**.** *A function $F : \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^\ell$, for $\ell = \ell(k) = \mathsf{poly}(k)$, is pseudo-random if for all* PPT *adversaries $\mathcal{A}$,*

$$\left|\Pr\left[\mathcal{A}^{F_K(\cdot)}(1^k) = 1\right] - \Pr\left[\mathcal{A}^{f(\cdot)}(1^k) = 1\right]\right| \leq \mathsf{negl}(k),$$

where $K \xleftarrow{\$} \{0,1\}^k$ and f is chosen uniformly at random from the set of functions from $\{0,1\}^\ell$ to $\{0,1\}^\ell$.

There are a few things to notice about this definition. First, we give the adversary *oracle* access to F_K and f. This is denoted by $\mathcal{A}^{F_K(\cdot)}$ and $\mathcal{A}^{f(\cdot)}$ and simply means that we allow the adversary to make any polynomially-bounded number of queries to these functions before it returns its output. In particular, giving $\mathcal{A}$ oracle access to F_K means that it can query the function without seeing the key K (this is important for security) and giving it oracle access to f means it never has to store or evaluate f.

We can use efficiently-computable PRFs for any application where we would want to use random functions. Of course PRFs are not random functions, they only appear to be to polynomially-bounded algorithms. But in the asymptotic framework we already assume all our adversaries are polynomially-bounded so this is not a problem.

Pseudo-random permutations. A pseudo-random permutation (PRP) is a PRF that is bijective. A PRP is efficient if the permutation and its inverse can both be evaluated in polynomial-time. For PRPs we sometimes are interested in a stronger notion of security than Definition 5.3. In particular, we need the permutation to be indistinguishable from a random permutation when the adversary has oracle access to both the permutation and its inverse.

Definition 5.3 (Strong pseudo-random permutation)**.** *A function $P : \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^\ell$, for $\ell = \ell(k) = \mathsf{poly}(k)$, is strongly pseudo-random if for all* PPT *adversaries $\mathcal{A}$,*

$$\left|\Pr\left[\mathcal{A}^{P_K(\cdot),P_K^{-1}(\cdot)}(1^k) = 1\right] - \Pr\left[\mathcal{A}^{f(\cdot),f^{-1}(\cdot)}(1^k) = 1\right]\right| \leq \mathsf{negl}(k),$$

where $K \xleftarrow{\$} \{0,1\}^k$ and f is chosen uniformly at random from the set of permutations over $\{0,1\}^\ell$.

A note on instantiations. Concrete instantiations of PRFs include HMAC-SHA256 as well as various constructions based on number-theoretic and lattice problems. Concrete instantiations of PRPs include block ciphers like AES but they can also be constructed from any PRF using a construction known as the Feistel network. Of course, we cannot prove that a given block cipher is a PRP but for certain ciphers like AES we believe this is a reasonable assumption.

6 Proofs of Security

The last step in the provable security paradigm is to prove that the concrete construction we are analyzing meets the security definition. As an example, we give a proof that a simple

Let $F : \{0,1\}^k \times \{0,1\}^\ell \to \{0,1\}^\ell$ be a pseudo-random function. Consider the secret-key encryption scheme $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ defined as follows:

- $\mathsf{Gen}(1^k)$: sample and output $K \xleftarrow{\$} \{0,1\}^k$;
- $\mathsf{Enc}(K, m)$: sample $r \xleftarrow{\$} \{0,1\}^k$ and output $\mathrm{ct} := \langle r, F_K(r) \oplus m \rangle$;
- $\mathsf{Dec}(K, \mathrm{ct})$: parse ct as $\langle \mathrm{ct}_1, \mathrm{ct}_2 \rangle$ and output $m := F_K(\mathrm{ct}_1) \oplus \mathrm{ct}_2$.

Figure 1: The standard secret-key encryption scheme.

secret-key encryption scheme based on PRFs is CPA-secure. This scheme is sometimes called the *standard* secret-key encryption scheme and it is described in Fig. 1.

Theorem 6.1. *If F is pseudo-random, then SKE is CPA-secure.*

Proof. We show that if there exists a PPT adversary $\mathcal{A}$ that breaks the CPA-security of SKE then there exists a PPT adversary $\mathcal{B}$ that breaks the pseudo-randomness of F. More precisely, we will describe an adversary $\mathcal{B}$ that can distinguish between oracle access to F_K (for a uniform K) and a random function f as long as $\mathcal{A}$ wins in a $\mathbf{CPA}_{\mathsf{SKE},\mathcal{A}}(k)$ experiment (with non-negligible probability over $1/2$). In particular, $\mathcal{B}$ will leverage $\mathcal{A}$'s ability to break SKE to in turn break the pseudo-randomness of F.

The way we accomplish this is by having $\mathcal{B}$ simulate a CPA experiment for $\mathcal{A}$ and cleverly embed its own challenge—which here is to distinguish between F_K and f—in the challenge for $\mathcal{A}$—which here is to guess b with non-negligible probability over $1/2$. Note that when we do this, we have to be very careful to make sure that the CPA experiment we simulate is indistinguishable from a real one; otherwise we have no guarantee that $\mathcal{A}$ will be able to win with the right probability. One way to think of this is that if we don't simulate the experiment exactly and $\mathcal{A}$ can tell, then it can always refuse to output anything.

With this in mind, we now describe $\mathcal{B}$. Recall that it has oracle access to a function g which is either F_K or f and it needs to distinguish between these two cases. $\mathcal{B}$ first simulates a $\mathbf{CPA}_{\mathsf{SKE},\mathcal{A}}(k)$ experiment. Whenever it receives an encryption oracle query m from $\mathcal{A}$, it samples a random string $r_m \xleftarrow{\$} \{0,1\}^k$, queries its own oracle g on r_m and returns $\langle r_m, g(r_m) \oplus m \rangle$ to $\mathcal{A}$. Upon receiving the challenge messages m_0 and m_1 from $\mathcal{A}$, it samples a bit $b \xleftarrow{\$} \{0,1\}$ and a string $r \xleftarrow{\$} \{0,1\}^k$. It then queries its oracle g on r and returns $\langle r, g(r) \oplus m_b \rangle$ to $\mathcal{A}$. When it receives more encryption oracle queries from $\mathcal{A}$ it answers them as above. At the end of the experiment, $\mathcal{A}$ returns a bit b'. If $b' = b$, $\mathcal{B}$ outputs 1 otherwise it outputs 0. Intuitively, if $\mathcal{A}$ is able to guess the bit correctly, $\mathcal{B}$ guesses that it had oracle access to the pseudo-random function F_K and if $\mathcal{A}$ guesses the bit incorrectly then $\mathcal{B}$ guesses that it had oracle access to the random function f.

Let's analyze the probability that $\mathcal{B}$ can distinguish between F_K and f. We have,

$$\Pr\left[\, \mathcal{B}^{F_K(\cdot)} = 1 \,\right] = \Pr\left[\, \mathbf{CPA}_{\mathsf{SKE},\mathcal{A}}(k) = 1 \,\right] = \frac{1}{2} + \varepsilon(k), \tag{1}$$

where $\varepsilon(k)$ is non-negligible. The first equality holds by construction of $\mathcal{B}$ since: (1) it outputs 1 if and only if $\mathcal{A}$ guesses b correctly; and (2) when $\mathcal{B}$ has oracle access to F_K, the experiment it simulates for $\mathcal{A}$ is exactly a $\mathbf{CPA}_{\mathsf{SKE},\mathcal{A}}(k)$ experiment. The second equality holds by our initial hypothesis about $\mathcal{A}$ (i.e., that it breaks the CPA-security of SKE).

In the following claim, we analyze the probability that $\mathcal{B}$ outputs 1 when given oracle access to a random function.

Claim. $\Pr\left[\mathcal{B}^{f(\cdot)}=1\right] \leq 1/2 + q/2^k$, where q is the number of queries $\mathcal{A}$ makes to its encryption oracle.

Let $\widetilde{\mathsf{SKE}} = (\widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$ be the same as SKE with the exception that the pseudo-random function F is replaced with a random function f. That is, $\widetilde{\mathsf{Gen}}$ simply outputs a random function and $\widetilde{\mathsf{Enc}}$ and $\widetilde{\mathsf{Dec}}$ use f in place of $F_{\mathcal{K}}$. Let reuse be the event in $\mathbf{CPA}_{\widetilde{\mathsf{SKE}},\mathcal{A}}(k)$ that at least one the random strings r used in the encryption oracle queries is used to generate the challenge ciphertext ct. Clearly, we have

$$\begin{aligned}
\Pr\left[\mathcal{B}^{f(\cdot)}=1\right] &= \Pr\left[\mathbf{CPA}_{\widetilde{\mathsf{SKE}},\mathcal{A}}(k)=1\right] \\
&= \Pr\left[\mathbf{CPA}_{\widetilde{\mathsf{SKE}},\mathcal{A}}(k)=1 | \mathsf{reuse}\right] \cdot \Pr\left[\mathsf{reuse}\right] + \\
&\quad \Pr\left[\mathbf{CPA}_{\widetilde{\mathsf{SKE}},\mathcal{A}}(k)=1 | \overline{\mathsf{reuse}}\right] \cdot \Pr\left[\overline{\mathsf{reuse}}\right] \\
&\leq \Pr\left[\mathsf{reuse}\right] + \Pr\left[\mathbf{CPA}_{\widetilde{\mathsf{SKE}},\mathcal{A}}(k)=1 | \overline{\mathsf{reuse}}\right]. \qquad (2)
\end{aligned}$$

The first equality follows by construction of $\mathcal{B}$ since: (1) $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ guesses b correctly; and (2) when $\mathcal{B}$ has oracle access to f, the experiment it simulates for $\mathcal{A}$ is exactly a $\mathbf{CPA}_{\widetilde{\mathsf{SKE}},\mathcal{A}}(k)$ experiment. We now bound both terms of Eq. (2). If q is the number of queries $\mathcal{A}$ makes to its encryption oracles we have,

$$\Pr\left[\mathsf{reuse}\right] = \Pr\left[\bigvee_{i=1}^{q} \mathsf{reuse}_i\right] \leq \sum_{i=1}^{q} \Pr\left[\mathsf{reuse}_i\right] \leq \sum_{i=1}^{q} \frac{1}{2^k} \leq \frac{q}{2^k},$$

where reuse_i is the event that the randomness used in the challenge is the same as the randomness used in $\mathcal{A}$'s ith encryption oracle query, where the first inequality follows from the union bound, and the second inequality follows from the fact that r is chosen uniformly at random. Finally, if reuse does not occur, the challenge ciphertext $\mathrm{ct} := \langle r, f(r) \oplus m_b \rangle$ is generated with completely new randomness and, therefore, ct is a uniformly distributed string (since f is a random function). The best $\mathcal{A}$ can do to guess b in this case is to just guess at random. So we have,

$$\Pr\left[\mathbf{CPA}_{\widetilde{\mathsf{SKE}},\mathcal{A}}(k)=1\right] \leq \frac{1}{2}.$$

□

We can now finish the proof. In particular, we have from Eq. (1) and the Claim above that,

$$\left|\Pr\left[\mathcal{B}^{F_K(\cdot)}=1\right]-\Pr\left[\mathcal{B}^{f(\cdot)}=1\right]\right| \geq \frac{1}{2}+\varepsilon(k)-\frac{1}{2}-\frac{q}{2^k} \geq \varepsilon(k)-\frac{q}{2^k}.$$

However, since $\mathcal{A}$ is polynomially-bounded it follows that it can make at most a polynomial number of queries. We therefore have that $q=\mathsf{poly}(k)$ and that $\varepsilon(k)-q/2^k$ is non-negligible, which is a contradiction.

■

7 Limitations of Provable Security

The provable security paradigm is the standard way of analyzing cryptographic primitives and protocols. Today, a well-designed cryptosystem is expected to come with a proof of security. As central as this paradigm is, however, it is important to keep in mind some of its limitations.

Definitions. A proof of security is only as meaningful as the security definition it is trying to meet. If the adversarial model captured by the definition does not correspond to real-world adversaries then the proof has little value. Because of this it is crucial to really understand how primitives are used in practice. This understanding is what allows us to formulate security definitions that provide meaningful guarantees.

Assumptions. The security of most cryptosystems relies on some underlying assumptions. These can be computational assumptions about number-theoretic or algebraic problems (e.g., factoring is hard, finding the shortest vector in a lattice is hard), or assumptions about certain primitives (e.g., AES is a PRP). So most of the time, when you see a statement in an Introduction or Abstract that says "*protocol X is provably-secure*" you should understand that there is usually an underlying assumption somewhere that the author is not making explicit. [5] The reason authors don't always state assumptions explicitly is for ease of exposition or because the assumption is considered standard (e.g., factoring is hard or AES is a PRP) but you should always be aware of what the underlying assumptions are when you are working with a cryptosystem. In addition, any Theorem about the security of a primitive or protocol should clearly state what the assumptions are. In time you should also develop an intuition about which assumptions are reasonable and which are less reasonable.

Errors in proofs. Unfortunately, there will occasionally be errors in proofs. Sometimes the error in a proof is fatal and the construction is not secure. In other cases the error can be fixed and the security of the scheme stands. Just be aware of this.

[5] Note that in some cases, the protocols are *information-theoretically* secure which means that they do not rely on assumptions.

8 Asymptotic vs. Concrete Security

The asymptotic framework used here makes analysis easier but has important limitations in practice. In particular, it does not allow us to set the parameters of our schemes because proofs in the asymptotic framework only tell us that the schemes are secure for large enough k but they do not help us determine what is large enough. In practice, of course, we actually need to choose concrete values for k so that we can use the schemes.

But let's take a closer look at a typical reduction from a security proof. Let Σ be some cryptographic scheme and Π be its underlying assumption. A proof of security for Σ based on Π would then have the form,

> *If an adversary can break* Σ *in time* t *with probability at least* ε_Σ*, then there exists an adversary that can break* Π *in time* t' *with probability at least* ε_Π*,*

where $t' \approx t$. To get the contradiction we usually have that

$$\varepsilon_\Pi \geq \varepsilon_\Sigma - \gamma$$

where ε_Σ is assumed to be non-negligible (in k) and $0 \leq \gamma \leq 1$ is shown to be negligible (in k). But note that if we had a precise expression for γ (as a function of t) then we would have a bound,

$$\varepsilon_\Sigma \leq \varepsilon_\Pi + \gamma, \tag{3}$$

that we could use as follows. Suppose we want to set the parameters of Σ such that $\varepsilon_\Sigma \leq 2^{-k}$. It follows by Eq. (3) that the parameters of Π need to be set such that,

$$\varepsilon_\Pi = 2^{-k} - \gamma.$$

You can see from this that we need to decrease the adversary's success probability against Π by γ and make the primitive *more* secure. But this means we have to increase its security parameter which has the effect of decreasing the efficiency of both Π and Σ. In particular, this implies that the term γ is very important as it has an effect on how we parameterize our construction and on its efficiency. Security proofs with a precise analysis of γ are referred to as *concrete* and reductions with small γ's are referred to as *tight*.