Security Models Notes

Security Models
MSc Program at UBa Cyber Crypto Center 2025
Jorge L. Villar

Last updated: Oct 16 17:43:25 2025

Contents

1 Security Models

In modern cryptography, security of cryptosystems is addressed in a formal way under what is known as **provable security**. This formalization uses precise definitions of security notions as well as a detailed specification of the algorithms and protocols that conform a cryptosystem.

(Generated by lineprocx v2.97)

Security is analyzed as an asymptotic property, as a function of a complexity parameter λ (also known as "security parameter"), that is usually related to the size of the cryptographic keys involved in the system.

1.1 Hard and Easy Tasks: The Complexity Class BPP

The first notion that needs to be formalized is the meaning of a problem or a task to be easy or hard. The tasks to be carried out by honest parties must be easy enough, what is normally understood as tasks that can be solved in polynomial time by probabilistic algorithms, with overwhelming probability (typically, $1 - \mathbf{negl}(\lambda)$).

The tasks that are considered as attacks must be infeasible for an adversary with a reasonable amount of resources. The complexity class BPP is used as the class of problems that are feasible to an attacker.

A problem family is called **decisional** if the solution to any of its instances is "yes" or "no" (or simply, a single bit). If the problem instance is specified as a binary string, then the problem family can be modelled as a subset L_1 of $\{0,1\}^*$, called a language, consisting of all the encodings of the yes-instances of the problem family P. For simplicity, we assume that any binary string not in the language is considered as a no-instance.

Definition 1 *A decision problem family P is in BPP if for any probabilistic polynomial time algorithm B*

- *The probability that B outputs 1 on any yes-instance is greater than 2/3.*
- *The probability that B outputs 1 on any no-instance is less than 1/3.*

The two thresholds can be widely modified without changing the meaning of the definition. For instance, instead of 2/3 and 1/3, they could be $1/2 + 1/Q(\lambda)$ and $1/2 - 1/Q(\lambda)$ for any arbitrary polynomial Q. Similarly, they can also be replaced by $1 - 2^{-\lambda}$ and $2^{-\lambda}$.

We say that a problem is infeasible for an adversary if it is not in the class BPP.

1.2 Worst Case Complexity vs. Average Case

The most commonly used complexity classes (P, NP, BPP, ...) are based on a worst-case analysis: the complexity of a problem family is defined by its hardest instances, no matter how many instances are hard enough. However, in cryptography we are interested in the average-case complexity, because secret keys and other elements in a cryptosystem are chosen at random. Therefore, the possible easy instances in a problem family must be bounded to a negligible fraction of the family. Then, the probability distribution of the yes-instances and the no-instances of a problem must be specified.

Definition 2 *A decisional problem family is considered hard with respect to given probability distributions of yes- and no-instances, $D_{1,\lambda}$ and $D_{0,\lambda}$, if for any probabilistic polynomial time algorithm B*

$|Prob(B(x) = 1 | x \leftarrow_{\$} D_{1,\lambda}) - Prob(B(x) = 1 | x \leftarrow_{\$} D_{0,\lambda})| \in \boldsymbol{negl}(\lambda)$

*In this case, we say that the two distributions $D_{1,\lambda}$ and $D_{0,\lambda}$ are **polynomially indistinguishable**, and the above probability difference is called the **advantage** of B in distinguishing both distributions.*

Example 1 (The Decisional Diffie Hellman Problem (DDH)) *Given an **instance generator** algorithm I that on the input of λ it generates a cyclic group G of prime order q, where q is λ bits long, together with a generator g of the group, the DDH problem associated to I is the following:*

Given (G, q, g) obtained from $I(\lambda)$, choose a random bit $b \in \{0,1\}$ and sample a tuple x from the probability distribution D_b, where

$D_{1,\lambda} = (G, q, g, g^u, g^v, g^{uv})$ *for uniformly distributed $u, v \in Z_q$,*

$D_{0,\lambda} = (G, q, g, g^u, g^v, g^w)$ *for uniformly distributed $u, v, w \in Z_q$.*

The problem is to guess the bit b only from the tuple x.

*The **DDH Assumption** associated to the instance generator I says that the two probability distributions $D_{0\lambda}$, $D_{1\lambda}$ are polynomially indintinguishable.*

Some attack goals can be modelled with decisional problem families that fit in the previous definition. However, in some other tasks the attacker is supposed to find a large object instead of producing just one bit. In these cases, the associated problem family is called a **search** problem family (instead of decisional).

Definition 3 *A search problem family is considered hard with respect to a given probability distribution D_λ of the instances if for any probabilistic polynomial time algorithm B*

$Prob(B(x) \in Sol(x) | x \leftarrow_\$ D_\lambda) \in \boldsymbol{negl}(\lambda)$

where $Sol(x)$ is the set of solutions of the problem instance x.

*The previous probability is called the **success probability** of B, and it is sometimes also called the **advantage** of B.*

Example 2 (The Computational Diffie Hellman Problem (CDH)) *Using the same instance generator I as in DDH, the CDH problem associated to I is the following:*

Given (G, q, g) obtained from $I(\lambda)$, and the group elements g^u, g^v, for uniformly distributed $u, v \in Z_q$, compute g^{uv}.

*The **CDH Assumption** associated to the instance generator I says that the success probability of any probabilistic polynomial time in solving the CDH problem is a negligible function in λ.*

If $\mathrm{Sol}(x)$ has more than one element (for most problem instances) the search problem family is sometimes called **flexible**.

Example 3 (A flexible version of CDH) *Using the same instance generator I as in DDH, given (G, q, g) obtained from $I(\lambda)$, and the group elements g^u, g^v, for uniformly distributed $u, v \in Z_q$, compute a pair (h, h^{uv}) for any generator h of G.*

1.3 Reductions and Security Proofs

In complexity theory the hardness of two different problem families can often be related via what is called a **reduction**. A reduction R from the problem family P_1 to another problem family P_2 is a transformation that when applied to a probabilistic polynomial time algorithm B_2 successfully solving P_2, it gives another probabilistic polynomial time algorithm $B_1 = R[B_2]$ that successfuly solves P_1.

When a reduction R from P_1 to P_2 exists we say that P_1 is reducible to P_2, and we denote this as $P_1 \Rightarrow P_2$. Notice that the previous reducibility assertion proves the hardness implication "P_1 is hard implies that P_2 is also hard". Assume otherwise that P_1 is hard but P_2 is not. Then, there would exist a probabilistic polynomial time algorithm B_2 successfully solving P_2. Therefore, $R[B_2]$ would be also a probabilistic polynomial time algorithm successfully solving P_1, which would be a contradiction.

The statement $P_1 \Rightarrow P_2$ reads both as "the problem P_1 is not harder that P_2'' and "'P_1 is hard' is not a weaker assumption than 'P_2 is hard"'.

In a typical reduction, R is an **oracle** algorithm that receives an instance of P_1, then it generates one or more instances of P_2 that are submitted to one or several instances of B_2, and finally, it uses the answers given by these instances to built its own answer to the received P_1 instance.

The reduction works if:

1. For any B_2 running in polynomial time, $R[B_2]$ also runs in polynomial time.
2. For any B_2 with a non-negligible advantage solving P_2, $R[B_2]$ also has a non-negligible advantage solving P_1,

A reduction is called **Black-Box** if it only uses B_2 as an oracle, that is, is has no access to the "internals" of B_2 and it only uses its outputs for conveniently selected inputs. We usually write $R[B_2] = R^{B_2}$ to denote this oracle access.

Example 4 (DDH $\Rightarrow$ CDH) *Let us show an explicit black-box reduction R that uses an algorithm B solving CDH problem to solve DDH problem.*

R receives as input the tuple $x = (G, q, g, U, V, W)$. Then it calls B on the input of (G, q, g, U, V) obtaining some group element W'. If $W = W'$ then R outputs 1. Otherwise, R outputs 0.

The running times of B and R^B are nearly the same (the time associated to a comparison or to the handling of the input and output parameters is usually negligible compared to performing the nontrivial task of solving CDH).

To analyze the advantage of R^B, first consider the case $b = 1$ (that is, there are uniformly distributed $u, v \in Z_q$ such that $U = g^u$, $V = g^v$ and $W = g^{uv}$). Whenever B gives the right answer, $W' = g^{uv}$, so does R^B, because $W = W'$ and then the output given by R^B is exactly b. Therefore,

$Prob(R^B(x) = 1 | x \leftarrow_{\$} D_{1,\lambda}) \geq Succ^B_{CDH}(\lambda)$,

where $Succ^B_{CDH}(\lambda)$ is the success probability of B solving CDH.

On the other hand, for $b = 0$ there are uniformly distributed $u, v, w \in Z_q$ such that $U = g^u$, $V = g^v$ and $W = g^w$. Therefore, W is a uniformly distributed group element that is independent of U, V, and hence it is also independent to the answer W' given by B. Thus,

$Prob(R^B(x) = 1 | x \leftarrow_{\$} D_{0,\lambda}) = 1/q$,

and

$Adv^{R^B}_{DDH}(\lambda) \geq |Succ^B_{CDH}(\lambda) - 1/q|$

where $Adv^{R^B}_{DDH}(\lambda)$ is the advantage of R^B solving DDH.

Observe that $1/q$ is of the order of $2^{-\lambda}$ and then it is a negligible function. Therefore, a non-negligible $Succ^B_{CDH}(\lambda)$ also implies a non-negligible $Adv^{R^B}_{DDH}(\lambda)$.

The previous reduction is called **tight** because R^B performs as well as B in terms of running time and advantage.

Example 5 (flex-CDH $\Rightarrow$ CDH) *In this case, there is a trivial black-box tight reduction because both CDH and flex-CDH instances have identical descriptions, and the solution to CDH contains a solution to flex-CDH. Namely, if W is a valid solution to the CDH instance $x = (G, q, g, U, V)$, then (g, W) is a valid solution to the corresponding flex-CDH instance.*

There is no other known reduction relating the three problems DDH, CDH and flex-CDH. Notice than DDH gives only a single bit, and then a reduction from CDH to DDH would require solving nearly λ DDH instances to find a solution to a CDH instance. On the other hand, an algorithm solving flex-CDH has total freedom to choose the generator h. Therefore, it seems very hard to translate the pair (h, h^{uv}) into g^{uv} without knowing the discrete logarithm of h respect to g.

In the provable security paradigm, the claim "a cryptosystem is secure against some specific attack" is encoded as the hardness of certain problem family capturing the meaning of the attack. Since absolute security proofs are not known and they are not expected to be found, a security proof is always relative to some extra assumptions. Thus, a security proof is a reduction from a problem family P conjectured to be hard to a problem family associated to a specific attack to a specific cryptosystem.

Example 6 ("ElGamal key recovery $\Rightarrow$ Discrete log problem easy") *Lets assume that a key recovery attack consists of giving a honestly generated public key to an attacker who then obtains the secret key from it.*

Formally, the attacker B receives an instance (G, q, g, y), where y is a random group element, and then it outputs $x \in Z_q$ such that $y = g^x$.

But thus us exactly the description of the discrete logaritm problem on (G, q, g). Therefore, the reduction is trivial since the two problem families are exactly the same.

1.4 Defining Attacks with Games

A formal definition of a security notion usually involves the detailed description of a specific attack, including the goal of the attacker and the resources it has access to. These resources can include the interaction of the attacker with other parties running the cryptographic protocol, that consists of different communication rounds. As a consequence, the problem families associated to security notions have a description far more complex than the simple examples given above (CDH, DDH. ...).

The usual way to define a security notion as a problem family is to specify the interaction of the attacker to the rest of the world via a random experiment performed by the attacker itself and a challenger (simulating the rest of the world). The challenger starts the conversation giving some input to the attacker. Then, the attacker sequentially queries the challenger to obtain some extra information, and eventually, the attacker gathers all the information received and ends the experiment giving some output to the challenger. Finally, the challenger processes all the information including the attacker's output and then it decides its own output, that is "yes" or "no" (or 1 or 0) indicating the success of the attacker.

This experiment is usually called a **security game**.

Example 7 (Key Recovery security game) *The game is played between a challenger Ch and an attacker A for a public key encryption scheme $PKE = (KeyGen(), Enc(), Dec())$*

- *Ch runs $(pk, sk) = KeyGen(\lambda)$*
- *Ch sends pk to A*
- *A outputs sk'*
- *Ch outputs 1 if $sk' = sk$, or 0, otherwise.*

We will describe the security games formally as pseudocode run by the challenger, which sends and receives information from the attacker A.

```
Game KeyRec(PKE, A, λ) {
  (pk, sk) ← PKE.KeyGen(λ)
  sk' ← A(pk)
  if sk' = sk output 1
  else output 0
  endif
}
```

$\text{Adv}_{\text{KeyRec}}(\text{PKE}, A, \lambda) = \text{Prob}[\text{KeyRec}(\text{PKE}, A, \lambda) = 1]$

Formally, we say that PKE is secure against KeyRec attacks if for any probabilistic polynomial time attacker A, $\text{Adv}_{\text{KeyRec}}(\text{PKE}, A, \lambda) \in \mathbf{negl}(\lambda)$. We will denote this assertion as KeyRec(PKE).

2 Security Models for Encryption

The previous security game example captures a vey basic attack against a public key encryption scheme. In this section, more realistic attacks are formally defined.

2.1 One-Wayness

One-wayness of a public key encryption scheme refers to the infeasibility of inverting the encryption function without knowing the secret key. It is formalized in the following game:

```
Game OW-CPA(PKE, A, λ) {
  (pk, sk) ← PKE.KeyGen(λ)
  m ←$ PKE.M_pk
  c* ← PKE.Enc(pk, m)
  m' ← A(pk, c*)
  if m' = m output 1
  else output 0
  endif
}
```

$\text{Adv}_{\text{OW-CPA}}(\text{PKE}, A, \lambda) = \text{Prob}[\text{OW-CPA}(\text{PKE}, A, \lambda) = 1]$

In this game the attacker receives just the public key and the encryption of a random message that it tries to guess. Obviously, if the attacker is able to launch a successful key recovery attack, then it easily wins the game by decrypting c_* with the learned secret key. However, there could exist other ways to decrypt a particular ciphertext without learning the secret key.

The name 'OW-CPA' comes from "One-Wayness under a Chosen Plaitext Attack", meaning that the attacker has only access to the encryption function and then it can observe ciphertexts corresponding to messages of its choice. This is the minimal set of capabilities an attacker can have in a public key encryption scheme, in contrast to what happens with a symmetric key encryption scheme, where the encryption functionality depends on the secret key and it is not accesible to the attacker.

2.2 Semantic Security

In the previous security notion an attacker still could recover half of the message without breaking the one-wayness of the public key encryption scheme. Semantic security is a stronger security notion that requires the attacker to be unable to learn any useful information about the message from the ciphertext, other than the information that it could directly get without it (e.g., the length of the message).

This notion is quite hard to formalize. However, it has been shown to be equivalent to the notion of indistinguishability of ciphertexts, formally described below:

```
Game IND-CPA(PKE, (A_0, A_1), λ) {
  (pk, sk) ← PKE.KeyGen(λ)
  (m_0, m_1, st) ← A_0(pk)
  if m_0 ∉ PKE.M_pk or m_1 ∉ PKE.M_pk output 0
  endif
  b ←$ {0, 1}
  c* ← PKE.Enc(pk, m_b)
  b' ← A_1(c*, st)
  if b'=b output 1
  else output 0
  endif
}
```

$\text{Adv}_{\text{IND-CPA}}(\text{PKE}, (A_0, A_1), \lambda) = |2\text{Prob}[\text{IND-CPA}(\text{PKE}, (A_0, A_1), \lambda) = 1] - 1|$

Here, the attacker A is split into two different stages (A_0, A_1). The first stage receives and processes the public key and outputs two carefully selected (valid) messages in the message space for the given public key. The first attacker stage communicates with the second one by means of a binary string st, which format and length depend entirely on the attacker. The string st can be seen as a representation of the internal

state of the first stage (e.g., including the public key), that will be used to initialize the second stage. This communication using states is the common model to formalize a stateful entity that is sequentially interacting with other entities.

The second stage of the attacker receives (in addition to the state information st) a ciphertext of one of the two selected messages. Then it tries to guess which one has been encrypted.

The indistinguishability game guarantees that even when the attacker knows everything about the message except for a single bit of information, it has no noticeable advantage compared to a lazy attacker that justs flips a coin to decide the missing message bit.

Observe that if an attacker is able to successfully launch a OW-CPA attack, then it will win the IND-CPA game. Indeed, the attacker can use the OW-CPA attack to learn the message m_b from the ciphertext c, and then infer b comparing m_b to the two messages selected in the first stage.

On the other hand, the IND-CPA security notion is unachievable by any deterministic encryption scheme like RSA. Indeed, the attacker can trivially win the game by just comparing the received ciphertext to the unique encryption of m_0 or m_1.

2.3 Active and Adaptive Attacks

In all previous attack models the attacker only eavesdrops at the insecure communication channel between the sender and the receiver, learning public keys and ciphertexts. However, some practical attacks are based on the injection of malicious messages or ciphertexts so that the attacker can get crucial information from the reactions of the communicating entities.

Active attacks consider the possibility of injection, replacement or removal of messages from the insecure communication channel, while adaptive attacks allow to split the attack into several rounds in a way that the attacker can decide its strategy according to the information received in previous steps.

In the case of public key encryption schemes, the most important active and adaptive attack is the **Chosen Ciphertext Attack (CCA)**. Namely, the adversary can sequentially submit ciphertexts (well-formed or not) do the decryption function and learn the message contained on them (or the failure or rejection of some badly-formed ciphertexts). This ability is formally modelled as a **decryption oracle** given to the attacker.

The access to the decryption oracle can happen before or after receiving the challenge (that is, the encryption of a random message, in the case of one-wayness, or the encryption of one of the two selected messages, in the case of indistinguishability). To define meaningful security notions, the decryption oracle access after receiving the challenge must be limited. Otherwise, if the attacker is allowed to query the oracle with the challenge ciphertext, it will surely win the security game.

We give below the security games defining the chosen ciphertext security versions of one-wayness and ciphertext indistinguishability:

```
Game OW-CCA1(PKE, (A_0, A_1), λ) {
  (pk, sk) ← PKE.KeyGen(λ)
  st ← A_0^{D_0}(pk)
  m ←$ PKE.M_pk
  c_* ← PKE.Enc(pk, m)
  m' ← A_1(c_*, st)
  if m' = m output 1
  else output 0
  endif
}

Oracle D_0(c) {
  output PKE.Dec(pk, sk, c)
}
```

$\mathrm{Adv}_{\text{OW-CCA1}}(\mathrm{PKE}, (A_0, A_1), \lambda) = \mathrm{Prob}[\text{OW-CCA1}(\mathrm{PKE}, (A_0, A_1), \lambda) = 1]$

```
Game OW-CCA2(PKE, (A_0, A_1), λ) {
  (pk, sk) ← PKE.KeyGen(λ)
  st ← A_0^{D_0}(pk)
  m ←$ PKE.M_pk
  c_* ← PKE.Enc(pk, m)
  m' ← A_1^{D_1}(c_*, st)
  if m' = m output 1
  else output 0
  endif
}

Oracle D_0(c) {
  output PKE.Dec(pk, sk, c)
}

Oracle D_1(c) {
  if c = c_* reject
  else output PKE.Dec(pk, sk, c)
  endif
}
```

$\mathrm{Adv}_{\text{OW-CCA2}}(\mathrm{PKE}, (A_0, A_1), \lambda) = \mathrm{Prob}[\text{OW-CCA2}(\mathrm{PKE}, (A_0, A_1), \lambda) = 1]$

```
Game IND-CCA1(PKE, (A_0, A_1), λ) {
  (pk, sk) ← PKE.KeyGen(λ)
  (m_0, m_1, st) ← A_0^{D_0}(pk)
  if m_0 ∉ PKE.M_pk or m_1 ∉ PKE.M_pk output 0
  endif
  b ←$ {0, 1}
  c_* ← PKE.Enc(pk, m_b)
  b' ← A_1(c_*, st)
  if b' = b output 1
  else output 0
  endif
}

Oracle D_0(c) {
  output PKE.Dec(pk, sk, c)
}
```

$\mathrm{Adv}_{\text{IND-CCA1}}(\mathrm{PKE}, (A_0, A_1), \lambda) = |2\mathrm{Prob}[\text{IND-CCA1}(\mathrm{PKE}, (A_0, A_1), \lambda) = 1] - 1|$

```
Game IND-CCA2(PKE, (A_0, A_1), λ) {
  (pk, sk) ← PKE.KeyGen(λ)
  (m_0, m_1, st) ← A_0^{D_0}(pk)
  if m_0 ∉ PKE.M_pk or m_1 ∉ PKE.M_pk output 0
  endif
  b ←$ {0, 1}
  c_* ← PKE.Enc(pk, m_b)
  b' ← A_1^{D_1}(c_*, st)
  if b' = b output 1
  else output 0
```

```
  endif
}

Oracle D_0(c) {
  output PKE.Dec(pk, sk, c)
}

Oracle D_1(c) {
  if c = c_* reject
  else output PKE.Dec(pk, sk, c)
  endif
}
```

$\text{Adv}_{\text{IND-CCA2}}(\text{PKE}, (A_0, A_1), \lambda) = |2\text{Prob}[\text{IND-CCA2}(\text{PKE}, (A_0, A_1), \lambda) = 1] - 1|$

IND-CCA1 is informally called "lunchtime attack" because the attacker eventually gains access to the decryption functionality (but not to the secret key) at some point before receiving the challenge ciphertext. IND-CCA2 is normally referred as IND-CCA, and it is considered nowadays as the standard security level for a general purpose public key encryption scheme.

2.4 Implications

We summarize here some of the generic implications between security notions mentioned before and also add some new ones.

OW-CCA2(PKE) $\Rightarrow$ OW-CCA1(PKE) $\Rightarrow$ OW-CPA(PKE),

where OW-CCA2(PKE) means the assertion "PKE is secure against a OW-CCA2(PKE) attack", or equivalently, $\text{Adv}_{\text{OW-CCA2}}(\text{PKE}, (A_0, A_1), \lambda) \in \mathbf{negl}(\lambda)$ for all probabilistic polynomial time algorithms A_0 and A_1.

The leftmost implication comes from the fact that an attacker that wins the OW-CCA1 game also wins the OW-CCA2 by just ignoring the decryption oracle in the second stage. Similarly, for the rightmost implication, an attacker A capable of winning the OW-CPA game can be transformed into a trivial first stage A_0 that ignores the decryption oracle and sets $st = pk$, and the second stage is just $A_1 = A$.

IND-CCA2(PKE) $\Rightarrow$ IND-CCA1(PKE) $\Rightarrow$ IND-CPA(PKE).

These implications are proved in the same way, letting the attackers ignore the extra decryption oracle provided in the game.

IND-atk(PKE) $\Rightarrow$ OW-atk(PKE) for any atk$\in \{\text{CPA}, \text{CCA1}, \text{CCA2}\}$.

Here we are assuming that the message space has a super-polynomial size (otherwise, OW security cannot be achieved). In any of these implications, an attacker (A_0, A_1) winning the OW-atk game can be transformed (in a black-box way) into another (B_0, B_1) winning the IND-atk game. Indeed, B_0 runs $st_A \leftarrow A_0^{D_0}(pk)$ (or just sets st_A to the empty string if atk=CPA), then chooses two random different messages $m_0, m_1 \in \text{PKE.M}_{pk}$, and outputs (m_0, m_1, st), where $st = (st_A, m_0, m_1)$. In the second stage, B_1 runs A_1 (or A, if atk=CPA) with the same oracles it received, using as inputs the received st_A and the challenge c_*. Then it compares A's output m' to m_1. It they are equal than B_1 outputs 1. Otherwise, it outputs 0.

This reduction can be expressed more precisely writting it as pseudocode. For instance, for atk=CCA2:

```
B_0^{D_0}(pk) {
  st_A ← A_0^{simD_0}(pk)
  m_0 ←$ PKE.M_pk
  m_1 ←$ PKE.M_pk \ {m_0}
  st = (m_0, m_1, st_A)
  output (m_0, m_1, st)
```

```
}

Oracle simD_0(c) {
  output D_0(c)
}

B_1^{D_1}(c_*, st) {
  parse st = (m_0, m_1, st_A)
  m' ← A_1^{simD_1}(c_*, st_A)
  if m' = m_1 output 1
  else if m' = m_0 output 0
  else output a random bit
  endif
}

Oracle simD_1(c) {
  output D_1(c)
}
```

Observe that the reduction (B_0, B_1) has to answer all oracle queries made by (A_0, A_1). But in this case, the simulated oracles given to (A_0, A_1) behave identically as the real oracles given to (B_0, B_1). Hence, the simulated oracles simply forward the queries and answers to and from the real oracles.

No deterministic encryption can achieve IND-CPA security (and then, neither IND-CCA1 nor IND-CCA2), but it is still possible that some of them achieve OW-CCA2 security.

None of the basic encryption schemes described in the last chapter can resist IND-CCA2 attacks or even OW-CCA2 attacks. For instance, an attacker against OW-CCA2(RSA) can compute $c = 2^e c_* \bmod n$, and submit c to the decryption oracle D_1. Since $c \neq c_*$, the oracle will return $2m \bmod n$, thus revealing the message m to the attacker.

Another example is ElGamal encryption. From the challenge ciphertext $c_* = (g^r, y^r m)$, the attacker computes a different ciphertext of the same message $c = c_*(g, y) = (g^{r+1}, y^{r+1} m)$. Then, the decryption oracle will directly give the message m to the adversary.

Achieving IND-CCA2 security was a very challenging task that required decades of research to come up with some realistic solutions (specially, for security proofs in the Standard Model, that is, whithout assuming the existence of Random Oracles).

3 Security Models for Signatures

Security models for digital signatures are defined in a similar way as it is done for public key encryption schemes. However, the security goals in digital signatures are very different, compared to encryption schemes: Here, the attacker must be unable to forge any signature that has not been generated by the legitimate signer. More precisely, the attacker must be unable to find new solutions to the signature verification equation without the knowledge of the signing key.

3.1 Universal Unforgeability

In a universal unforgeability (UF) scenario the attacker must be unable to find a valid signature for a random message not chosen by itself. The minimal attacker capabilities are a key-only attack (KOA), in which the attacker only knows the public (verification) key. In a more realistic attack, the attacker has also access to a list of valid pairs message/signature for randomly selected messages (RMA).

More formally, assuming a digital signature scheme DS = (KeyGen(), Sig(), Ver()):

```
Game UF-KOA(DS, A, λ) {
  (pk, sk) ← DS.KeyGen(λ)
  m* ←$ SK.M_pk
  (s*) ← A(pk, m*)
  output DS.Ver(pk, m*, s*)
}
```

$\mathrm{Adv}_{\text{UF-KOA}}(\mathrm{DS}, A, \lambda) = \mathrm{Prob}[\text{UF-KOA}(\mathrm{DS}, A, \lambda) = 1]$

```
Game UF-RMA(DS, A, λ) {
  (pk, sk) ← DS.KeyGen(λ)
  m* ←$ SK.M_pk
  (s*) ← A^S(pk, m*)
  output DS.Ver(pk, m*, s*)
}

Oracle S() {
  m ←$ SK.M_pk \ {m*}
  output (m, DS.Sig(sk, m))
}
```

$\mathrm{Adv}_{\text{UF-RMA}}(\mathrm{DS}, A, \lambda) = \mathrm{Prob}[\text{UF-RMA}(\mathrm{DS}, A, \lambda) = 1]$

3.2 Existential Unforgeability

In the existential unforgeability (EF) scenario the attacker wins the game if it is able to find any valid pair message/signature for a message of its choice, but not signed before.

```
Game EF-KOA(DS, A, λ) {
  (pk, sk) ← DS.KeyGen(λ)
  (m*, s*) ← A(pk)
  output DS.Ver(pk, m*, s*)
}
```

$\mathrm{Adv}_{\text{EF-KOA}}(\mathrm{DS}, A, \lambda) = \mathrm{Prob}[\text{EF-KOA}(\mathrm{DS}, A, \lambda) = 1]$

```
Game EF-RMA(DS, A, λ) {
  lm ← List.Empty
  (pk, sk) ← DS.KeyGen(λ)
  (m*, s*) ← A^S(pk)
  if DS.Ver(pk, m*, s*) and m* ∉ lm
    output 1
  else output 0
  endif
}

Oracle S() {
  m ←$ SK.M_pk
  lm.AddToList(m)
  output (m, DS.Sig(sk, m))
}
```

$\mathrm{Adv}_{\text{EF-RMA}}(\mathrm{DS}, A, \lambda) = \mathrm{Prob}[\text{EF-RMA}(\mathrm{DS}, A, \lambda) = 1]$

In the last game, the oracle is "stateful", that is it maintains an internal state consisting of the list lm of messages queried to the oracle. This is necessary to allow the challenger to check the freshness of the attacker's forgery in the last step of the game.

There is a stronger version of EF-RMA in which a forgery (m_*, s_*) for $m_* \in lm$ is still accepted as valid unless it is exactly one of the pairs returned by the oracle. Namely, forging a second signature for a message for which a signature is known is considered a successful attack.

Notice that if a signature scheme is deterministic (i.e., there exists a unique signature for a given message and a given public key) there is no difference between the stronger and the plain versions of EF-RMA.

In order to write a formal game for the strong EF-RMA, the list lm is replaced by a list lms, storing the message/signature pairs returned by the oracle.

3.3 Active and Adaptive Attacks

As in the case of encryption schemes, in a realistic scenario an attacker could inject some messages and persuade the signer to produce a valid signature on it. This model is called Chosen Message Attack (CMA). See below the corresponding security games for the two goals UF and EF:

```
Game UF-CMA(DS, A, λ) {
  lm ← List.Empty
  (pk, sk) ← DS.KeyGen(λ)
  m_* ←$ SK.M_pk
  s_* ← A^S(pk, m_*)
  output DS.Ver(pk, m_*, s_*) and m_* ∉ lm
}

Oracle S(m) {
  lm.AddToList(m)
  output DS.Sig(sk, m)
}
```

$\mathrm{Adv}_{\text{UF-CMA}}(\mathrm{DS}, A, \lambda) = \mathrm{Prob}[\text{UF-CMA}(\mathrm{DS}, A, \lambda) = 1]$

```
Game EF-CMA(DS, A, λ) {
  lm ← List.Empty
  (pk, sk) ← DS.KeyGen(λ)
  (m_*, s_*) ← A^S(pk)
  if DS.Ver(pk, m_*, s_*) and m_* ∉ lm
    output 1
  else output 0
  endif
}

Oracle S(m) {
  lm.AddToList(m)
  output DS.Sig(sk, m)
}
```

$\mathrm{Adv}_{\text{EF-CMA}}(\mathrm{DS}, A, \lambda) = \mathrm{Prob}[\text{EF-CMA}(\mathrm{DS}, A, \lambda) = 1]$

As in the case of EF-RMA, there is a strong version of EF-CMA defined in the same way.

EF-CMA (or the strong version of it) is the commonly used standard level of security for a general purpose digital signature scheme.

3.4 Implications

There exist similar generic implications between security notions for signatures to the case of encryption schemes. Namely, CMA security implies RMA security, which in turn implies KOA security for both UF and EF goals.

In the first implication, the RMA oracle can be simulated calling the CMA oracle on randomly selected messages. For the second implication, the adversary simply ignores the oracle.

On the other hand, EF-atk security implies UF-atk security for any atk $\in \{\text{KOA}, \text{RMA}, \text{CMA}\}$. Indeed, any valid UF forgery (that is, a valid message/signature pair for the message selected by the challenger) can be used as a valid EF forgery. In particular, the EF adversary simulates the UF challenger by generating a random message and asking the UF adversary for a signature on it.

Moreover, the strong EF-atk security also implies the plain one, because any plain EF-atk forgery is also considered as a valid strong EF-atk forgery.

4 Security Proofs

In this section we describe some useful techniques for building security proofs. A security proof claiming that a particular cryptosystem achieves certain security level is often based on a reduction from a well-known problem family that is believed to be hard to the problem family associated to the formal game defining this security level.

A very easy example is proving that IND-CPA(ElGamal) is implied by the DDH Assumption. The proof consists of giving an explicit reduction from the DDH Problem to the IND-CPA(ElGamal) problem. The particular proof described below is generic, meaning that it is valid for any instance generator (i.e., for any way to choose the group G and its generator g). This does not mean thay IND-CPA(ElGamal) holds for any group family, but at least it holds whenever the DDH Assumption is true.

Let $I()$ be the instance generator used in ElGamal encryption, that on the input of $\lambda \in Z^+$ it outputs a tuple (G, q, g) where G is a cyclic group of prime order q, g is a generator of G and the bitlength of q is λ. The reduction is a black-box one, and it works as follows:

```
BBReduction B^(A0,A1)(G, q, g, U, V, W) {
  param ← (G, q, g)
  pk ← (param, U)
  (m0, m1, st) ← A0(pk)
  if m0 ∉ G or m1 ∉ G output 0
  endif
  b ←$ {0, 1}
  c* ← (V, W m_b)
  b' ← A1(c*, st)
  if b' = b output 1
  else output 0
  endif
}
```

Observe that $B^{(A_0,A_1)}$ is seen by the attacker (A_0, A_1) as its own challenger in the game IND-CPA(ElGamal), while it is also seen as the attacker in the following game formally defining DDH:

```
Game DDH(I, B, λ) {
  (G, q, g) ← I(λ)
  b ←$ {0, 1}
  u, v ← Z_q
```

$U \leftarrow g^u$
$V \leftarrow g^v$
if $b = 1$
$W \leftarrow g^{uv}$
else
$W \leftarrow_{\$} G$
endif
$b' \leftarrow B(G, q, g, U, V, W)$
if $b' = b$ **output** 1
else output 0
}

$\text{Adv}_{\text{DDH}}(I, B, \lambda) = |2\text{Prob}[\text{DDH}(I, B, \lambda) = 1] - 1|$

Sometimes it is useful to split Game DDH into two separate games: DDH-0 and DDH-1, where DDH-b means that b is preset to the given value instead of choosing it at random. Clearly,

$\text{Adv}_{\text{DDH}}(I, B, \lambda) = |\text{Prob}[\text{DDH-0}(I, B, \lambda) = 1] + \text{Prob}[\text{DDH-1}(I, B, \lambda) = 1] - 1|$

To proceed with the security proof by reduction, we start considering the case b=1, that is, we compute $\text{Prob}[\text{DDH-1}(I, B, \lambda)] = 1$.

Let us define the view of an attacker A in a security game G, denoted as $\text{View}_A(G(A, \lambda))$, as the sequence of random variables consisting of the inputs received from the challenger and the outputs sent by the attacker. When some extra oracles are given to the attacker, $\text{View}_A(G(A, \lambda))$ also includes the corresponding oracle queries and responses. We say that two games $G1$ and $G2$ are identical for an attacker A if $\text{View}_A(G1(A, \lambda))$ and $\text{View}_A(G2(A, \lambda))$ are identically distributed random variables.

In our case, the two views $\text{View}_{A_0,A_1}(\text{IND-CPA}(\text{ElGamal}, (A_0, A_1), \lambda)) = (G, q, g, y, m_0, m_1, st, c_*, b')$ and $\text{View}_{A_0,A_1}(\text{DDH-1}(I, B^{A_0,A_1}, \lambda)) = (G, q, g, U, m_0, m_1, st, c_*, b')$, that is the view of A in the composition of the DDH-1 challenger and the reduction B, are identically distributed. Indeed, (G, q, g) is drawn from $I(\lambda)$ in both games, and y or U are just uniformly distributed random group elements. Therefore, A_0 receives in both games identically distributed inputs, then so are its outputs (m_0, m_1, st).

Moving forward in the games, the challenge ciphertexts are generated identically, since the variables (V, W) in DDH-1 can be identified to (g^r, y^r), and the message encrypted in the ciphertext is selected in both games at random from the two messages produced by A_0.

As a consequence, A_1 also receives identically distributed inputs in both games (even conditioned to the previous components in the view), and then its output b' is also identically distributed in both games.

From the above considerations, we conclude that the event $b' = 1$ in Game $\text{DDH-1}(I, B^{A_0,A_1}, \lambda)$ and the event $b' = b$ in Game $\text{IND-CPA}(\text{ElGamal}, (A_0, A_1), \lambda)$ occur with the same probability, and then

$\text{Prob}[\text{DDH-1}(I, B^{A_0,A_1}, \lambda) = 1] = \text{Prob}[\text{IND-CPA}(\text{ElGamal}, (A_0, A_1), \lambda) = 1]$.

The previous analysis is often summarized as the assertion "the reduction B in Game DDH-1 perfectly simulates the Game IND-CPA(ElGamal) for A".

It remains the analysis of the game DDH-0, in which the behavior is dramatically different: The two views $\text{View}_{A_0,A_1}(\text{IND-CPA}(\text{ElGamal}, (A_0, A_1, \lambda))$ and $\text{View}_{A_0,A_1}(\text{DDH-0}(I, B^{A_0,A_1}, \lambda))$ have different probability distributions. Indeed, in the second game W is independent of $(G, q, g, U, m_0, m_1, st, V)$, and therefore the second component of c_* perfectly hides the encrypted message. Hence, it also perfectly hides the bit b used to select that message. This means that A_1 receives no information about this bit, and then

$\text{Prob}[\text{DDH-0}(I, B^{A_0,A_1}, \lambda) = 1] = 1/2$.

Summarizing the two results:

$\text{Adv}_{\text{DDH}}(I, B^{A_0,A_1}, \lambda) = |\text{Prob}[\text{DDH-0}(I, B^{A_0,A_1}, \lambda) = 1] + \text{Prob}[\text{DDH-1}(I, B^{A_0,A_1}, \lambda) = 1] - 1| =$
$= |\text{Prob}[\text{IND-CPA}(\text{ElGamal}, (A_0, A_1), \lambda) = 1] - 1/2| = \text{Adv}_{\text{IND-CPA}}(\text{ElGamal}, (A_0, A_1), \lambda)/2.$

This concludes the proof by reduction, because if DDH Assumption holds, then the above advantage must be negligible for any (A_0, A_1), what means that ElGamal encryption achieves IND-CPA security.

4.1 Game Sequences and the Difference Lemma

The last example of security proof is actually a very simple case: there are three games involved: DDH-0, DDH-1 and IND-CPA(ElGamal), and the proof consists of plugging DDH-1 into IND-CPA(ElGamal) in a way that shows that they are indeed the same game. Next, DDH-0 can be seen as a slight modification of DDH-1 (both games differ only in the way W is generated), but when replacing the copy of DDH-1 embedded into IND-CPA(ElGamal) by a copy of DDH-0, the resulting game gives advantage exactly 0 to any attacker (that is, perfect secrecy). As a consequence, any attack against IND-CPA(ElGamal) also serves to tell apart games DDH-0 and DDH-1, which would mean that DDH problem is easy.

This reasoning line can be rewritten as a sequence of games, $(\text{Game}_0, ..., \text{Game}_n)$, with the following properties:

- The first game in the sequence captures the security notion to be proved.
- The last game is a trivial game, in which any attacker has zero advantage.
- The "distance" between any two consecutive games in the sequence can be bounded.

From the triangle inequality, the distance between the first and the last games in the sequence is upper bounded by the sum of all intermediate distances, which is also an upper bound of the advantage of an attacker in the first game, since the last game in the sequence is trivial.

More formally,

$\text{Adv}_{\text{Game } n}(A, \lambda) = 0$

$\text{Adv}_{\text{Game } 0}(A, \lambda) \leq |\text{Adv}_{\text{Game } 1}(A, \lambda) - \text{Adv}_{\text{Game } 0}(A, \lambda)| + ... + |\text{Adv}_{\text{Game } n}(A, \lambda) - \text{Adv}_{\text{Game } n-1}(A, \lambda)|$

In the IND-CPA(ElGamal) example, the sequence has two games: the first one is IND-CPA(ElGamal) and the second one is a trivial game. The distance of the two games is upper bounded by twice the advantage of an attacker against DDH, which we assume to be a negligible function.

We can consider three types of game hops in a sequence of games:

1. A **trivial hop**: Games i and $i + 1$ are identical random experiments, that is, the views of A in both games are identically distributed. Thus,

   $\text{Adv}_{\text{Game } i}(A, \lambda) = \text{Adv}_{\text{Game } i+1}(A, \lambda)$.

   This step usually corresponds to a way to rewrite a game in a more convenient way for subsequent steps.

2. A **statistical hop**: Games i and $i + 1$ are almost identical. More precisely, there exist events F_i in Game i and F_{i+1} in Game $i + 1$ with identical and negligible probability p_F, such that the view of A in Game i conditioned to "F_i does not occur" is identically distributed to its view in Game $i + 1$ conditioned to "F_{i+1} does not occur". In this case,

   $|\text{Adv}_{\text{Game } i+1}(A, \lambda) - \text{Adv}_{\text{Game } i}(A, \lambda)| \leq p_F \in \mathbf{negl}(\lambda)$.

3. A **computational hop**: Any attacker A can be transformed into another one B solving a computational problem P with advantage bounded by a polynomial in $|\text{Adv}_{\text{Game } i+1}(A, \lambda) - \text{Adv}_{\text{Game } i}(A, \lambda)|$. In this case, we will need the assumption that the computational problem P is hard in order to conclude the security proof.

The result in the second type of game hop is based on a basic fact about random experiments, often called the Difference Lemma:

Lemma 1 (Difference Lemma) *Let us consider two random experiments Exp_1 and Exp_2 producing two probability distributions with a common support X. If there exist an event F (defined as a predicate on X) that occurs with probability p_F in both experiments, such that Exp_1 conditioned to "F does not occur" is identical to Exp_2 conditioned to "F does not occur", then for any event A (also defined as a predicate on X), $|Prob_{Exp_1}[A] - Prob_{Exp_2}[A]| \leq p_F$.*

Proof. $|\text{Prob}_{\text{Exp}_1}[A] - \text{Prob}_{\text{Exp}_2}[A]| = |\text{Prob}_{\text{Exp}_1}[A|F]\text{Prob}_{\text{Exp}_1}[F] - \text{Prob}_{\text{Exp}_2}[A|F]\text{Prob}_{\text{Exp}_2}[F]| =$
$= |(\text{Prob}_{\text{Exp}_1}[A|F] - \text{Prob}_{\text{Exp}_2}[A|F])|p_F \leq p_F$.

In the literature more sophisticated structures than a sequence of games are considered, but the key idea is providing a way that the initial security game is modified in small steps so that every change can be analyzed in isolation to the others, even if dozens of steps are needed to complete the proof.

4.2 Hybrid Arguments

The length of a sequence of games used in a security proof is not necessarily a fixed constant. Indeed, it can depend on some parameters of the cryptosystem analyzed.

A classical example is a generic reduction from the IND-CPA security of an encryption scheme to the IND-CPA security of the same encryption scheme used to encrypt vectors of n messages. For instance, one can use ElGamal to encrypt a vector $(m_1, ..., m_n)$ as $(g^{r_1}, y^{r_1}m_1, ..., g^{r_n}, y^{r_n}m_n)$. The question is how secure is encrypting vectors if we only know that ElGamal is secure for single messages.

There is a generic proof that works for any IND-CPA secure encryption scheme for single messages. In the IND-CPA security definition for tha n-fold version of the basic encryption scheme (with a single key for all the n components)

$\text{Enc}^n(pk, (m_1, ..., m_n)) = (\text{Enc}(pk, m_1), ..., \text{Enc}(pk, m_n))$

the attacker's goal is telling apart the encryptions of two vectors of n messages chosen by itself.

Consider the sequence of games (Game $0, ...,$ Game n) such that Game i is like the IND-CPA security game for the n-fold encryption scheme but the first i components of the vector of messages encrypted in the challenge ciphertext are chosen from the first vector given by the adversary, and the last $n - i$ are chosen from the second. In particular, the original IND-CPA game measures the difference between Games 0 and n.

The proof consists of a reduction from the IND-CPA security of the basic scheme to the difference of any two consecutive games in the sequence. As a consequence, the advantage of an attacker against IND-CPA of the n-fold encryption is at most n times the advantage of another attacker against the IND-CPA of the basic encryption scheme. This result gives an upper bound to the security degradation when the encrytion scheme is used to encrypt vectors instead of single elements.

Let us see how the reduction works. The attacker against IND-CPA of the basic scheme $(B_{n,i,0}, B_{n,i,1})$ uses the attacker (A_0, A_1) that tries to tell apart Game i and Game $i + 1$. The former attacker just uses the i-th copy of the basic scheme to make the latter decide the message encrypted in the challenge ciphertext.

```
BBReduction B^{(A0,A1)}_{n,i,0}(pk) {
  (m0, m1, st_A) ← A0(pk)
  for each j in {1, ..., i − 1}
    c_j ← Enc(pk, m_{0,j})
  endfor
  c_i ←⊥
  for each j in {i + 1, ..., n}
    c_j ← Enc(pk, m_{1,j})
  endfor
  st ← (st_A, c)
  output (m_{0,i}, m_{1,i}, st)
}

BBReduction B^{(A0,A1)}_{n,i,1}(c*, st) {
  parse st = (st_A, c)
  c_i ← c*
  b' ← A1(c, st_A)
```

```
  output b'
}
```

The simulation of Game i is perfect when $b = 0$ in the IND-CPA game, while Game $i + 1$ is perfectly simulated when $b = 1$ in the IND-CPA game.

The intermediate games are called **hybrid** games because they "interpolate" between the first and the last games in the sequence.

4.3 Random Self-Reducibility

In some cases, the last result given in the previous section can be improved and the proof works without the need if hybrid games. This is the case of ElGamal encryption. The trick is related to a property called **random self-reducibility**, that refers to the ability of transforming any arbitrary instance of a problem family to a uniformly distributed one with a probabilistic polynomial time algorithm. Diffie-Hellman problems (both the decisional and the computational ones) enjoy this property. For instance, in the decisional case, the following transformation:

$(G, q, g, U, V, W) \mapsto (G, q, g, U', V', W') = (G, q, g, U^{\alpha} g^{\beta}, V^{\gamma} g^{\delta}, W^{\alpha\gamma} U^{\alpha\delta} V^{\beta\gamma} g^{\beta\delta})$;
for $\alpha, \gamma \leftarrow_{\$} Z_q^{\times}$ and $\beta, \delta \leftarrow_{\$} Z_q$

maps any instance in the support of the distribution $D_{b,\lambda}$ (see Example 1 in Section 1.2) into a random instance following the distribution $D_{b,\lambda}$, for both $b = 0$ and $b = 1$. Indeed, if $U = g^u$, $V = g^v$, $W = g^w$, $U' = g^{u'}$, $V' = g^{v'}$ and $W' = g^{w'}$, we can write

$u' = \alpha u + \beta$
$v' = \gamma v + \delta$
$w' = u'v' + \alpha\gamma(w - uv)$

and then it is straightforward to show that (u', v', w') follows the right distribution for both values of b. Interestingly, (u', v', w') and (u, v, w) conditioned to $b = 0$ or $b = 1$ are independent random vectors.

This last observation makes possible to "amplify" the difference between the distributions $D_{0,\lambda}$ and $D_{1,\lambda}$ to their n-fold version with the same kind of transformation:

$(G, q, g, U, V, W) \mapsto (G, q, g, U_1, V_1, W_1, ..., U_n, V_n, W_n)$
where $(U_i, V_i, W_i) = (U^{\alpha_i} g^{\beta_i}, V^{\gamma_i} g^{\delta_i}, W^{\alpha_i\gamma_i} U^{\alpha_i\delta_i} V^{\beta_i\gamma_i} g^{\beta_i\delta_i})$; for $\alpha_i, \gamma_i \leftarrow_{\$} Z_q^{\times}$ and $\beta_i, \delta_i \leftarrow_{\$} Z_q$

This trick makes possible to prove the IND-CPA security of the n-fold version of ElGamal encryption with a constant number of games (i.e., a number of games independent of n). The idea is transforming a DDH challenge (U, V, W) into a challenge vector $(U_1, V_1, W_1, ..., U_n, V_n, W_n)$ and use the latter to produce a valid encryption of a message vector $\mathbf{m}$ given by the adversary (when the DDH tuple follows the distribution D_1) or a valid encryption of a random message vector independent of $\mathbf{m}$ (when the DDH tuple follows the distribution D_0). We actually need here a special case of the transformation in which all U_i are equal to U, that is, all $\alpha_i = 1$ and all $\beta_i = 0$. Therefore, the reduction works exactly in the same way as in the plain ElGamal encryption.

```
BBReduction B^(A0,A1)(G, q, g, U, V, W) {
  param ← (G, q, g)
  pk ← (param, U)
  (m0, m1, st) ← A0(pk)
  if m0 ∉ G^n or m1 ∉ G^n output 0
  endif
  b ←$ {0, 1}
  for each i in {1, ..., n}
    γi ←$ Zq^×
    δi ←$ Zq
```

$V_i \leftarrow V^{\gamma_i} g^{\delta_i}$
$W_i \leftarrow W^{\gamma_i} U^{\delta_i}$
$c_{*,i} \leftarrow (V_i, W_i m_{b,i})$
endfor
$b' \leftarrow A_1(c_*, st)$
if $b' = b$ **output** 1
else output 0
endif
}

With this reduction, we obtain
$\mathrm{Adv}_{\mathrm{DDH}}(I, B^{A_0,A_1}, \lambda) = \mathrm{Adv}_{\text{IND-CPA}}(\mathrm{ElGamal}^n, (A_0, A_1), \lambda)/2$.

4.4 Granularity

When formally defining a problem family there is some freedom to choose the notion of hardness. For instance, in the definition of DDH, a DDH instance is a tuple (G, q, g, U, V, W) drawn from one of two probability distributions that the attacker wants to tell apart. The freedom comes from the way we compute the attacker's advantage.

In the definition given in previous sections, this advantage is averaged among all problem instances that corresponds to a particular value of the parameter λ, and the instance probability distribution comes from the instance generator $I()$. Thus, even when DDH achieves this notion of hardness, there could exist a few weak problem instances (provided they occur with a small enough probability). We call this kind of definition a **high granularity** hardness notion.

Finer levels of granularity are achieved when we ask the advantage of the attacker to be negligible for any choice of the group G in the range of the instance generator (**medium granularity**), or even for every particular generator of the group (**low granularity**). More precisely, in the medium granularity setting the attacker's advantage function is defined for every value of λ as the maximum of the advantages for each possible choice of the group G in the range of $I(\lambda)$, but it is still averaged using the randomness contained in the generator and the components (U, V, W).

In the low granularity setting, the advantage is maximized for every possible choice of (G, q, g), and then the averaging is only made with respect of the randomness contained in the components (U, V, W).

Note that the finer the granularity, the stronger is the hardness assumption. If a problem family is believed to be hard in the low granularity setting, then the values of (G, q, g) can be standarized for each value of λ, and then they can be shared among many users without any security degradation (this is actually the case of ElGamal encryption). However, some known reductions between problems related to DDH does not preserve the generator g, and then they cannot directly be applied in this setting.

4.5 Dealing with Multiple Calls: Tightness

Consider the following variant (actually, a particularization) of the CDH Problem, called **Square in the Exponent (SE)** Problem:

Example 8 (The Square in the Exponent Problem (SE)) *Using the same instance generator $I()$ as in DDH, the SE problem associated to $I()$ is the following:*

Given (G, q, g) obtained from $I(\lambda)$, and the group element g^u, for a uniformly distributed $u \in Z_q$, compute g^{u^2}.

*The **SE Assumption** associated to the instance generator $I()$ says that the success probability of any probabilistic polynomial time in solving the SE problem is a negligible function in λ.*

There exist reductions between CDH and SE in either way, so that we can say that the two associated hardness assumptions are equivalent. We show below the two explicit reductions. The first one is a black-box reduction from SE to CDH:

```
BBReduction B^{A_CDH}(G, q, g, U) {
  α ←$ Z_q^×
  output A_CDH(G, q, g, U, U^α)
}
```

Observe that simply taking $\alpha = 1$ does not work, because then the CDH instance received by A_{CDH} is not properly distributed and hence, we do not know any bound of the attacker's advantage (it can be non-negligible on a negligible fraction of the instances without contradicting the CDH Assumption).

The other reduction is a bit more sophisticated:

```
BBReduction B^{A_SE}(G, q, g, U, V) {
  W_+ ← A_SE(G, q, g, UV)
  W_− ← A_SE(G, q, g, U/V)
  output (W_+/W_−)^{1/4}
}
```

The analysis of the reduction if far more subtle, because the reduction calls twice the attacker A_{SE}, and it expects that the two calls result in right answers. Therefore, we need two facts:

1. The instance received in each call is properly distributed.
2. We can find a lower bound of the probability that in the two calls, both answers are correct.

The first condition clearly holds, because UV and U/V are uniformly distributed group elements, conditioned to the instance (G, q, g). The problem arises in the second requirement. In the low granularity setting, (G, q, g) are fixed and then we can show that UV and U/V are independent random variables. Therefore, the two outcomes corresponding to the calls giving the right answer are independent, and we are done. However, this approach does not directly work in the higher granularity levels, because both SE instances are correlated (they share the group and the generator). Hence we cannot claim independence in this case.

There are two ways to solve the problem: The first one is to randomize the second instance, replacing it by $(G, q, g^\alpha, (U/V)^\alpha)$ and raising the the second answer to the inverse of α mod q. This recovers the independence in the medium granularity setting. The second solution is finding a lower bound of the probability that the two answers are correct. It can be shown that, in the worst case, this probability equals that of the independence case:

Lemma 2 *Let Exp be a random experiment producing a random vector (X, Y), and let A be an event related to (X, Y), that is a predicate on the support of (X, Y). Consider now two independent realizations of the same experiment, producing vectors (X_1, Y_1) and (X_2, Y_2) and let A_1 and A_2 be the events that A respectively occurs in the first and the second realization of the experiment. Then*

Prob$[A_1$ *and* $A_2 | X_1 = X_2] \geq (Prob[A])^2$.

Proof. Define $p_x = \mathrm{Prob}[X = x]$ and $a_x = \mathrm{Prob}[A|X = x]$, for any x in the support of X. Clearly, $\mathrm{Prob}[A] = \sum_{x \in X} a_x p_x$ and $\mathrm{Prob}[A_1 \text{ and } A_2 | X_1 = X_2] = \sum_{x \in X} (a_x)^2 p_x$.

Consider now the dot product defined by the weight vector $(p_x)_{x \in X}$, that is $\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{x \in X} p_x u_x v_x$.

Then, defining the vectors $\mathbf{a} = (a_x)_{x \in X}$ and $\mathbf{1} = (1)_{x \in X}$, we can write
$\mathrm{Prob}[A] = \langle \mathbf{a}, \mathbf{1} \rangle$ and
$\mathrm{Prob}[A_1 \text{ and } A_2 | X_1 = X_2] = \langle \mathbf{a}, \mathbf{a} \rangle$.

Thus, by Cauchy-Schwartz inequality,
$(\text{Prob}[A])^2 = (\langle \mathbf{a}, \mathbf{1} \rangle)^2 \leq \langle \mathbf{a}, \mathbf{a} \rangle \langle \mathbf{1}, \mathbf{1} \rangle = \text{Prob}[A_1 \text{ and } A_2 | X_1 = X_2]$.

This lemma solves the previous issues at all granularity levels, and it is a very useful tool when dealing with several calls to an attacker where their inputs can be split into a common part and an independent part.

There is still a problem with these reductions calling the attacker more than once: the advantage of the reduction normally depends on a power of the attacker's advantage. This means that the advantage of the reduction would be much less that the one of the attacker, to the point that can make the reduction meaningless in practice. This is the reason why people is interested mainly in tight reductions, where the two advantages and the two execution times are similar (e.g., they only differ in a small multiplicative constant).

4.6 The Random Oracle Model

The Random Oracle Model (ROM) was created as a formal model for secure hash functions, identifying them with truly random functions. This model is very convenient to build security proofs of very efficient and versatile cryptosystems. However, it is only an idealized model because the definition of a realization of a random function needs exponential space, and it cannot be handled by polynomial time algorithms. As a consequence, the so-called "security proofs" in the ROM are nothing more than heuristic security arguments, but they are usually interpreted as an evidence that any attack against a cryptosystem must use an exploit of the corresponding hash function. In this case, the attack could be fixed by just replacing the hash function by a more robust one. To be fair, for most encryption or signature schemes used in real applications only security proofs in the ROM are known.

Even when truly random functions cannot be used by efficient algorithms, they can be efficiently simulated in a security proof. In fact, the view that a polynomial time algorithm (e.g., an attacker) have on a random function consists only of polynomially many evaluations. Therefore, a reduction can easily maintaing a table of random values, giving a perfect simulation of the random function. Namely, a table containing pairs $(x, H(x))$ for a random function $H : \{0,1\}^* \to Y$, where Y is a finite set, can be simulated in the following way:

- Start the simulation with an empty table.
- For any new point x, check whether it is stored in the table.
- If true, retrieve the known value of $H(x)$ and output it.
- Otherwise, choose a random element $y \in Y$ and store the pair (x, y) in the table. Then, output y.

As an example, we show below how ROM can be used to prove the Hashed ElGamal security, IND-CPA(HElGamal), under the CDH Assuption, which is believed to be strictly weaker than DDH. We start a sequence of games with Game 0, that is identical to the IND-CPA game except that the hash function of Hashed ElGamal encryption is now modelled as a random oracle, simulated by the challenger.

```
Game 0((A_0, A_1), λ) {
  t ← EmptyTable()
  (G, q, g) ← I(λ)
  x ←$ Z_q
  y ← g^x
  pk ← ((G, q, g), y)
  (m_0, m_1, st) ← A_0^{simH}(pk)
  r ←$ Z_q
  b ←$ {0, 1}
  c_* ← (g^r, simH(y^r) ⊕ m_b)
  b' ← A_1^{simH}(c_*, st)
```

```
  if b' = b output 1
  else output 0
  endif
}

Oracle simH(w) {
  if t.IsKeyInTable(w)
    k ← t.GetValue(w)
  else
    k ←$ {0,1}^λ
    t.AddToTable(w, k)
  endif
  output k
}
```

Observe that the challenger also uses the simulated random oracle $simH$ to build the challenge ciphertext. Otherwise, the simulation could be incorrect.

The next game, Game 1, is identical to the previous one from the point of view of the attacker. The main difference is the way the challenge ciphertext is generated.

```
Game 1((A_0, A_1), λ) {
  t ← EmptyTable()
  (G, q, g) ← I(λ)
  x ←$ Z_q
  U ← g^x
  V ←$ G
  W ← V^x
  pk ← ((G, q, g), U)
  (m_0, m_1, st) ← A_0^{simH}(pk)
  b ←$ {0,1}
  c_* ← (V, simH(W) ⊕ m_b)
  b' ← A_1^{simH}(c_*, st)
  if b' = b output 1
  else output 0
  endif
}

Oracle simH(w) {
  if t.IsKeyInTable(w)
    k ← t.GetValue(w)
  else
    k ←$ {0,1}^λ
    t.AddToTable(w, k)
  endif
  output k
}
```

Then,

$|\mathrm{Adv}_{\text{Game 1}}((A_0, A_1), \lambda) - \mathrm{Adv}_{\text{Game 0}}((A_0, A_1), \lambda)| = 0.$

The next game uses a random binary string in the challenge ciphertext, not extracted from $simH$.

```
Game 2((A_0, A_1), λ) {
  t ← EmptyTable()
  (G, q, g) ← I(λ)
  x ←$ Z_q
  U ← g^x
  V ←$ G
  k_* ←$ {0,1}^λ
  pk ← ((G, q, g), U)
  (m_0, m_1, st) ← A_0^{simH}(pk)
  b ←$ {0,1}
  c_* ← (V, k_* ⊕ m_b)
  b' ← A_1^{simH}(c_*, st)
  if b' = b output 1
  else output 0
  endif
}

Oracle simH(w) {
  if t.IsKeyInTable(w)
    k ← t.GetValue(w)
  else
    k ←$ {0,1}^λ
    t.AddToTable(w, k)
  endif
  output k
}
```

Consider the event $F_{1,2}$ that V^x is in the table t when the game ends, which is equivalent to say that at some point the attacker queried V^x to $simH$. Both games are identical if $F_{1,2}$ does not occur (since in this case there is no contradiction in the simulation of $simH$). Then we can apply the Difference Lemma to show that

$|\mathrm{Adv}_{\text{Game 2}}((A_0, A_1), \lambda) - \mathrm{Adv}_{\text{Game 1}}((A_0, A_1), \lambda)| \leq \mathrm{Prob}[F_{1,2}]$.

However, in Game 2 the bit b is completely hidden to the attacker, because it is only used in the expression $k_* \oplus m_b$. So b is completely masked by k_*, that is also used only there. Indeed, the views of the attacker conditioned to $b = 0$ and $b = 1$ are identically distributed. As a consequence, Game 2 is trivial, and $\mathrm{Adv}_{\text{Game 2}}((A_0, A_1), \lambda) = 0$.

Summing up,

$\mathrm{Adv}_{\text{IND-CPA}}(\text{HElGamal}^{\text{ROM}}, (A_0, A_1), \lambda) = \mathrm{Adv}_{\text{Game 0}}((A_0, A_1), \lambda) = \mathrm{Adv}_{\text{Game 1}}((A_0, A_1), \lambda) \leq \mathrm{Prob}[F_{1,2}]$.

To finish the proof, we show a reduction that transforms the attacker (A_0, A_1) into another attacker B that solves a CDH problem instance with advantage at least a non-negligible fraction of $\mathrm{Prob}[F_{1,2}]$:

```
BBReduction B^{(A_0,A_1)}(G, q, g, U, V) {
  t ← EmptyTable()
  k_* ←$ {0,1}^λ
  pk ← ((G, q, g), U)
  (m_0, m_1, st) ← A_0^{simH}(pk)
  b ←$ {0,1}
  c_* ← (V, k_* ⊕ m_b)
  b' ← A_1^{simH}(c_*, st)
  output t.GetARandomEntryValue
}
```

```
Oracle simH(w) {
  if t.IsKeyInTable(w)
    k ← t.GetValue(w)
  else
    k ←$ {0,1}^λ
    t.AddToTable(w, k)
  endif
  output k
}
```

The method t.GetARandomEntryValue selects at random one of the entries of the table t and outputs the stored value. The composition of the CDH game and the reduction B perfectly simulates Game 2 from the attacker's point of view. Notice that in Game 2, the secret key is only needed to generate the random group element U, that now is given as part of the CDH Problem instance.

Thus, conditioned to $F_{1,2}$, the CDH Problem instance solution is one of the entries in the table, and then the reduction succeeds with a probability at least $\text{Prob}[F_{1,2}]/Q_H$, where Q_H is the number of queries to $simH$ made by the attacker. In conclusion, we have shown that

$\text{Adv}_{\text{IND-CPA}}(\text{HElGamal}^{\text{ROM}}, (A_0, A_1), \lambda) \leq \text{Prob}[F_{1,2}] \leq Q_H Adv_{\text{CDH}}(B^{A_0,A_1}, \lambda)$.

This inequality ends the proof, because if CDH Assumption holds then $\text{Adv}_{\text{CDH}}(B^{A_0,A_1}, \lambda)$ is negligible, as Q_H is bounded by a polynomial. Then, $\text{Adv}_{\text{IND-CPA}}(\text{HElGamal}^{\text{ROM}}, (A_0, A_1), \lambda)$ is also negligible for any attacker.

Notice that the proof is far from being tight, because in practice a real attacker can perform a considerable amount of hash computations (e.g., Q_H could be as high as 2^{30}), which is the loss factor of the reduction.

4.7 The Generic Group Model

Another idealized analysis tool is the so-called **Generic Group Model**, in which the notion of group is abstracted to the point that an attacker can access the group functionality (e.g., neutral element, group operation and inverses) via some oracles. The goal is to analyze cryptosystems based on groups (like all cryptosystems related to the hardness of the Discrete Logarithm Problem) in a generic way, discarding any specific attack related to any particular representation of the group elements.

The abstract model behind the Generic Group Model is a map from a group G, say of prime order q, (used in the real algorithm) to the polynomial ring $Z_q[X_1, ..., X_n]$, where X_i is a formal variable that replaces one of the group elements in the input of the attacker's algorithm. For instance, if two group elements assigned to polynomials P_1 and P_2, are operated with the group operation, then the result is mapped to the polynomial $P_1 + P_2$. The formal variables $X_1, ..., X_n$ could also be considered as polynomials in other formal parameters $T_1, ..., T_d$ to make possible to encode in the polynomial ring some existing relations between the inputs of the attacker (e.g., if the inputs are taken from a DDH tuple).

This model is very limiting, but most of the known attacks to group-based cryptosystems are either avoidable by choosing a suitable group, or they fit well into the Generic Group Model. Therefore, this model can be seen as a formal tool that rules out any possible design mistake affecting the security of a group-based cryptosystem.

The main benefit of the model is translating a problem from complexity theory to a purely algebraic problem, and security proofs in the model often reduces to upper bounding the probability of finding polynomials of bounded degree vanishing at certain secret values.

4.8 Negative Results and Meta-Reductions

In the previous sections we only addressed (positive) security results, where we show things like "If a problem family P is hard, then the cryptosystem S is secure in the security model M". Other useful results in cryptography are the negative ones, which try to rule out the existence of some classes of reductions, like "there does not exist any black-box reduction in the Generic Group Model from problem A to problem B".

The usual technique to obtain negative results is to build an explicit meta-reduction, that is, a transformation that converts any reduction (fulfilling some additional constraints) between two given problems into a successful attacker that solves a third problem (that can be the same as one of the other two).

Sometimes negative results are used to find lower bounds on the sizes of objects, the number of necessary queries or the loss factor of a reduction, for a restricted but still meaningful class of reductions (like black-box reductions, or key-preserving reductions). Typically, most known reductions fit into these restricted classes.
