Introduction to Pairing-Based Cryptography

Mihir Bellare[1]

April 11, 2006

[1] Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093, USA. mihir@cs.ucsd.edu, http://www-cse.ucsd.edu/users/mihir

Preface

This is a set of notes for a class I am teaching on pairing based cryptography. The first version of the class was in Fall 04 and this version is Spring 06.

Thanks to Tom Ristenpart and Scott Yilek for comments.

Mihir Bellare San Diego, California USA

Contents

Chapter 1

Pairing basics

1.1 Basics

Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be groups of prime order p. Note since the order is prime, all these groups are cyclic. The last group is called the target group, whence the subscript. A *pairing* is an efficiently computable map $\mathbf{e}$: $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ satisfying the following conditions:

1. Bilinear: $\mathbf{e}(A_1B_1, A_2) = \mathbf{e}(A_1, A_2)\mathbf{e}(B_1, A_2)$ for every $(A_1, B_1) \in \mathbb{G}_1 \times \mathbb{G}_2$ and every $A_2 \in \mathbb{G}_T$, and $\mathbf{e}(A_1, A_2B_2) = \mathbf{e}(A_1, A_2)\mathbf{e}(A_{1,2})$ for every $A_1 \in \mathbb{G}_1$ and every $(A_2, B_2) \in \mathbb{G}_2$.

2. Non-degenerate: There exists $(A_1, A_2) \in \mathbb{G}_1 \times \mathbb{G}_2$ such that $\mathbf{e}(A_1, A_2) \neq 1$.

We use the notation 1 for the identity element, with the context hopefully making it clear in which group this is. We let $\mathrm{DLog}_g(A)$ denote the discrete logarithm of A to base generator g, the group being hopefully again clear from the context. We let $\mathbb{G}^*$ denote the set of non-identity elements of a group $\mathbb{G}$, which is the set of generators when, as here, the group being considered has prime order.

It is possible that $\mathbb{G}_1 = \mathbb{G}_2$. This is called the symmetric case or setting, while the asymmetric case or setting is the more general one in which $\mathbb{G}_1, \mathbb{G}_2$ are possibly different. As we will see, the asymmetric case allows uses of different groups that bring some performance benefits. We will try to work in the asymmetric setting as much as possible.

It is possible to fix an isomorphism ψ: $\mathbb{G}_2 \to \mathbb{G}_1$. In the symmetric case, this is the identity map. Note that if g_2 is a generator of $\mathbb{G}_2$ then $\psi(g_2)$ is a generator of $\mathbb{G}_1$.

The following is the central algebraic property of pairings that is responsible for a lot of their power.

Proposition 1.1 $\mathbf{e}(A_1^{a_1}, A_2^{a_2}) = \mathbf{e}(A_1, A_2)^{a_1 a_2}$ for any $(A_1, A_2) \in \mathbb{G}_1 \times \mathbb{G}_2$ and any $a_1, a_2 \in \mathbb{Z}$.

Proof: This follows easily by repeated applications of the bilinearity property. ▮

Proposition 1.2 If g_1 is a generator of $\mathbb{G}_1$ and g_2 is a generator of $\mathbb{G}_2$ then $\mathbf{e}(g_1, g_2)$ is a generator of $\mathbb{G}_T$.

Proof: Since $|\mathbb{G}_T|$ is prime, it suffices to show that $\mathbf{e}(g_1, g_2) \neq 1$. Suppose towards a contradiction that $\mathbf{e}(g_1, g_2) = 1$. Then for any $(A_1, A_2) \in \mathbb{G}_1 \times \mathbb{G}_2$ we have

$$\mathbf{e}(A_1, A_2) \;=\; \mathbf{e}(g_1^{a_1}, g_2^{a_2}) \;=\; \mathbf{e}(g_1, g_2)^{a_1 a_2} \;=\; 1^{a_1 a_2} \;=\; 1\ ,$$

where $a_i = \mathrm{DLog}_{g_i}(A_i)$ for $i = 1, 2$ and we used Proposition 1.1. This contradicts the fact that a pairing is non-degenerate. ▮

Exercise 1.3 Let $g_2 \in \mathbb{G}_2$. Let $f\colon\ \mathbb{G}_1 \to \mathbb{G}_T$ be the map defined by $f(h) = \mathbf{e}(h, g_2)$ for all $h \in \mathbb{G}_1$. Show that f is a group homomorphism, and an isomorphism if g_2 is a generator of $\mathbb{G}_2$. Formulate and show the analogous result for $g_1 \in \mathbb{G}_1$. ▮

Exercise 1.4 Let g_1 be a generator of $\mathbb{G}_1$ and g_2 a generator of $\mathbb{G}_2$. Both directly and as a consequence of Exercise 1.3, show that $\mathbf{e}(g_1, 1) = \mathbf{e}(1, g_2) = 1$. ▮

Is a pairing symmetric in the sense that $\mathbf{e}(A_1, A_2) = \mathbf{e}(A_2, A_1)$? In general, this equality does not even make sense since A_1, A_2 are in different groups. But in the symmetric setting it makes sense and is true. This is a consequence of the following more general fact, from which what we have just said follows by letting ψ be the identity function:

Proposition 1.5 Let $\psi\colon\ \mathbb{G}_2 \to \mathbb{G}_1$ be an isomorphism. Then $\mathbf{e}(\psi(A), B) = \mathbf{e}(\psi(B), A)$ for all $A, B \in \mathbb{G}_2$.

Proof: Let g_2 be a generator of $\mathbb{G}_2$. Let $a = \mathrm{DLog}_{g_2}(A)$ and $b = \mathrm{DLog}_{g_2}(B)$. Then

$$\begin{aligned}
\mathbf{e}(\psi(A), B) &= \mathbf{e}(\psi(g_2^a), g_2^b) \\
&= \mathbf{e}(\psi(g_2)^a, g_2^b) \\
&= \mathbf{e}(\psi(g_2), g_2)^{ab} \\
&= \mathbf{e}(\psi(g_2)^b, g_2^a) \\
&= \mathbf{e}(\psi(g_2^b), g_2^a) \\
&= \mathbf{e}(\psi(B), A)\ .
\end{aligned}$$

The second and fifth equalities used the fact that ψ is an isomorphism. The third and fourth equalities used Proposition 1.1. ▮

We say that (g_2, g_2^x, h, W) is a dh-tuple if $W = h^x$, where $g_2 \in \mathbb{G}_2^*$, $x \in \mathbb{Z}$ and $h, W \in \mathbb{G}_1$. The following says that such tuples can be easily recognized using the pairing.

Proposition 1.6 There is an efficiently computable function V such that $V(g_2, g_2^x, h, W)$ returns 1 if $W = h^x$ and 0 otherwise, for all $g_2 \in \mathbb{G}_2^*$, all $h, W \in \mathbb{G}_1$ and all $x \in \mathbb{Z}$.

Proof: The function V returns 1 if $\mathbf{e}(W, g_2) = \mathbf{e}(h, g_2^x)$ and 0 otherwise. To prove correctness, let g_1 be a generator of $\mathbb{G}_1$. Let $w = \mathrm{DLog}_{g_1}(W)$ and $\alpha = \mathrm{DLog}_{g_1}(h)$. Let $g = \mathbf{e}(g_1, g_2)$. Then $\mathbf{e}(W, g_2) = g^w$ and $\mathbf{e}(h, g_2^x) = g^{\alpha x}$ by Proposition 1.1. Proposition 1.2 tells us that g is a generator of $\mathbb{G}_T$, so $g^w = g^{\alpha x}$ iff $w \equiv \alpha x \pmod{p}$, meaning iff $W = h^x$. ▮

Exercise 1.7 Proposition 1.6 assumes $g_2 \in \mathbb{G}_2^*$. Is the statement still true when $g_2 = 1$? What does V return in this case when $W = h^x$, and what does it return when $W \neq h^x$? ▮

Problem	**Given**	**Determine**
DL($\mathbb{G}$)	$g \in \mathbb{G}^*;\ g^x \in \mathbb{G}$	x
CDH($\mathbb{G}$)	$g, h \in \mathbb{G}^*;\ g^x \in \mathbb{G}$	h^x
DDH($\mathbb{G}$)	$g, h \in \mathbb{G}^*;\ g^x, \mathrm{CH}_c(h^x) \in \mathbb{G}$	c
CDH	$g_2 \in \mathbb{G}_2^*;\ g_2^x \in \mathbb{G}_2;\ h \in \mathbb{G}_1^*$	h^x
DDH	$g_2 \in \mathbb{G}_2^*;\ g_2^x \in \mathbb{G}_2;\ h \in \mathbb{G}_1^*;\ \mathrm{CH}_c(h^x) \in \mathbb{G}_1$	c
BCDH	$g_1 \in \mathbb{G}_1^*;\ g_1^x, g_1^y \in \mathbb{G}_1;\ g_2 \in \mathbb{G}_2^*;\ g_2^x, g_2^z \in \mathbb{G}_2$	$\mathbf{e}(g_1, g_2)^{xyz}$
BDDH	$g_1 \in \mathbb{G}_1^*;\ g_1^x, g_1^y \in \mathbb{G}_1;\ g_2 \in \mathbb{G}_2^*;\ g_2^x, g_2^z \in \mathbb{G}_2;\ \mathrm{CH}_c(\mathbf{e}(g_1, g_2)^{xyz}) \in \mathbb{G}_t$	c
CLin	$u, v, h \in \mathbb{G}_1^*;\ u^x, v^y \in \mathbb{G}_1$	h^{x+y}
DLin	$u, v, h \in \mathbb{G}_1^*;\ u^x, v^y, \mathrm{CH}_c(h^{x+y}) \in \mathbb{G}_1$	c
q-SDH	$g_1 \in \mathbb{G}_1^*;\ g_2 \in \mathbb{G}_2^*;\ g_2^x, g_2^{x^2}, \ldots, g_2^{x^q} \in \mathbb{G}_2$	$(g_1^{1/(x+\alpha)}, \alpha)$

Figure 1.1: Some computational problems related to pairings. In q-SDH, q is a positive integer parameter of the problem, and the adversary has to determine a pair of the stated form for any $\alpha \in \mathbb{Z}_p^*$ of its choice.

1.2 Computational problems related to pairings

Pairing-based cryptography relies on assumptions about the hardness of various computational problems related to the groups and the pairing. There is in fact a slew of different problems in use. Here we will look at a few of the basic ones.

The problems are summarized in Fig. 1.1. Each problem is specified by saying what is given to the adversary and what the adversary has to determine to win. In the table, $\mathbb{G}$ denotes any of $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$. For the givens our convention is that any free variable is chosen at random from the set in which it is indicated to lie, with exponents chosen at random from $\mathbb{Z}_p$. For example, in CDH, we see $g_2, g_2^x \in \mathbb{G}_2$, $h \in \mathbb{G}_1$, which means g_2 is drawn at random in $\mathbb{G}_2$ and x is drawn at random from $\mathbb{Z}_p$. Now g_2^x is fixed (not free). Then h is drawn at random from $\mathbb{G}_1^*$. We denote by c a random challenge bit, and for $G \in \mathbb{G}$ and $\alpha \in \mathbb{Z}_p$ we let $\mathrm{CH}_c(G^\alpha)$ be G^α if $c = 1$ and a random element of $\mathbb{G}$ if $c = 0$. This is used to define decision problems.

If A is an adversary against a particular problem P then $\mathbf{Adv}^{\mathrm{P}}(A)$ denotes its advantage in solving this problem. This is the probability that A determines what the table asks it to determine when the probability is over the choices of the givens and the coins of A.

We say that a problem is easy if there is an efficient algorithm to solve it, and hard if there is no known efficient algorithm to solve it. These terms are informal but can be formalized in standard ways.

If $\mathrm{P}_1, \mathrm{P}_2$ are problems, then we write $\mathrm{P}_1 \rightarrow \mathrm{P}_2$ to mean that if P_1 is easy then P_2 is easy.

(Equivalently, if P_2 is hard then P_1 is hard.) We now discuss various things we know about the easiness of the problems defined and how they relate to each other via the $\rightarrow$ relation.

Our list of problems is far from exhaustive. You will find many others in the literature, and it is a good exercise to try to relate them to the ones here via the $\rightarrow$ relation.

Proposition 1.8 DDH is easy.

Proof: Follows from Proposition 1.6. ▮

Proposition 1.9 $\mathrm{DL}(\mathbb{G}_T) \rightarrow \mathrm{DL}(\mathbb{G}_2)$ and $\mathrm{DL}(\mathbb{G}_T) \rightarrow \mathrm{DL}(\mathbb{G}_1)$.

Proof: Let A_T be an algorithm that solves $\mathrm{DL}(\mathbb{G}_T)$. Consider algorithm A_2 that on input $g_2, g_2^x \in \mathbb{G}_2$ picks g_1 at random in $\mathbb{G}_1^*$, lets $g = \mathbf{e}(g_1, g_2)$ and $h = \mathbf{e}(g_1, g_2^x)$. Note that g is a generator of $\mathbb{G}_T$ by Proposition 1.2, and $h = g^x$ by Proposition 1.1. Now A_2 runs $A(g, h)$ and returns the result. A_2 solves $\mathrm{DL}(\mathbb{G}_2)$ and has the same advantage as A. The case of $\mathrm{DL}(\mathbb{G}_1)$ is analogous. ▮

Exercise 1.10 Convince yourself that the proof of Proposition 1.9 does not easily extend to show that $\mathrm{CDH}(\mathbb{G}_T) \rightarrow \mathrm{CDH}(\mathbb{G}_i)$ for $i = 1$ or $i = 2$. What does that tell you about whether or not these implications are true? ▮

Proposition 1.11 $\mathrm{DDH}(\mathbb{G}_T) \rightarrow \mathrm{DDH}(\mathbb{G}_2)$ and $\mathrm{DDH}(\mathbb{G}_T) \rightarrow \mathrm{DDH}(\mathbb{G}_1)$. ▮

Exercise 1.12 Prove Proposition 1.11. ▮

Proposition 1.13 If there exists an efficiently computable isomorphism ψ: $\mathbb{G}_2 \rightarrow \mathbb{G}_1$ then
(1) $\mathrm{DL}(\mathbb{G}_1) \rightarrow \mathrm{DL}(\mathbb{G}_2)$, and
(2) $\mathrm{DDH}(\mathbb{G}_2)$ is easy. ▮

Note even with the isomorphism, we are unable to relate $\mathrm{CDH}(\mathbb{G}_2)$ to $\mathrm{CDH}(\mathbb{G}_1)$. Also, we cannot conclude that $\mathrm{DDH}(\mathbb{G}_1)$ is easy.

Exercise 1.14 Prove Proposition 1.13. ▮

Proposition 1.15 $\mathrm{CDH}(\mathbb{G}_T) \rightarrow \mathrm{BCDH}$, $\mathrm{CDH}(\mathbb{G}_2) \rightarrow \mathrm{BCDH}$, $\mathrm{CDH}(\mathbb{G}_1) \rightarrow \mathrm{BCDH}$ and $\mathrm{CDH} \rightarrow \mathrm{BCDH}$.

Proof: Let A_T be an algorithm that solves $\mathrm{CDH}(\mathbb{G}_T)$. Consider algorithm A that on input $g_1, g_1^x, g_1^y, g_2, g_2^x, g_2^z$ lets $g = \mathbf{e}(g_1, g_2)$ and $h = \mathbf{e}(g_1^y, g_2^z)$. It runs A_T on $g, \mathbf{e}(g_1, g_2^x), h$ to get $h^x = \mathbf{e}(g_1, g_2)^{xyz}$, solving BCDH. Now let A_2 be an algorithm that solves $\mathrm{CDH}(\mathbb{G}_2)$. Consider algorithm A that on input $g_1, g_1^x, g_1^y, g_2, g_2^x, g_2^z$ runs $A_2(g_2, g_2^x, g_2^z)$ to obtain g_2^{xz} and then returns $\mathbf{e}(g_1^y, g_2^{xz})$, solving BCDH. The proofs that $\mathrm{CDH}(\mathbb{G}_1) \rightarrow \mathrm{BCDH}$ and $\mathrm{CDH} \rightarrow \mathrm{BCDH}$ are analogous and left as exercises. ▮

Proposition 1.16 $\mathrm{DDH}(\mathbb{G}_T) \rightarrow \mathrm{BDDH}$. ▮

Exercise 1.17 Prove Proposition 1.16. ▮

Recall that a family of functions F: $\mathsf{Keys} \times \mathsf{Dom} \to \mathsf{Range}$ is one-way if given K, y, where $K \xleftarrow{\$} \mathsf{Keys}$; $x \xleftarrow{\$} \mathsf{Dom}$; $y \leftarrow F(K, x)$, it is hard to find a (any) pre-image of y under $F(K, \cdot)$.

Exercise 1.18 Let F: $\mathbb{G}_2 \times \mathbb{G}_1 \to \mathbb{G}_T$ be defined by $F(g_2, h) = \mathbf{e}(h, g_2)$ for all $h \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. Show that F is one-way if CDH is hard. ▌

1.3 Finding pairings and cost issues

This is something I know very little about. I'd like to figure it out better. At the moment let me try to summarize a few things that I have heard. Say we want about 80 bits of security, meaning the discrete logarithm problems in all groups take at least 2^{80} steps. Let r_1, r_2, r_T denote the size (in bits) of the representation of an element in groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ respectively. Current estimates would give us, very roughly, the following:

Setting	$\lg(p)$	r_1	r_2	r_T
Symmetric	300	300	300	6*300
Asymmetric	160	160	3*160	6*160
Asymmetric with isomorphism	160	160	6*160	6*160

The difference between the second and third settings above is that in the latter we ask for an efficiently computable isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$. Notice that r_1 is smaller in the asymmetric settings, which is important for applications like short signatures.

The cost of computing the pairing is of the same order as an exponentiaion, meaning cubic, but varies depending on the groups, pairing and algorithm used. Estimates put it at anywhere between 4 and 20 times the cost of an exponentiation.

Chapter 2

Signatures

We begin with signatures one the one hand to illustrate some simple pairing based schemes that provide new properties like aggregation. But part of our motivation and goal here is to take a first step towards IBE by looking at the signature schemes underlying IBE schemes. This gets us to the heart of of the major IBE schemes in a simpler and more incremental way. Let us begin by recalling some signature basics.

2.1 Signature schemes and their security

Unless otherwise indicated, an algorithm may be randomized. A (digital) signature scheme $\mathsf{DS} = (\mathsf{KG}, \mathsf{SGN}, \mathsf{VF})$ is specified as usual by its key-generation, signing and verifying algorithms. Via $(pk, sk) \xleftarrow{\$} \mathsf{KG}$ a signer can generate a public and private key for itself. Via $S \leftarrow \mathsf{SGN}(sk, M)$ it can generate a signature σ of a message M. The deterministic verification algorithm VF takes pk, a message M and candidate signature S and returns 1 (accept) or 0 (reject) to indicate whether or not the signature is valid. We require the usual consistency, meaning $\mathsf{VF}(pk, M, \mathsf{SGN}(sk, M)) = 1$ with probability one for any M, the probability being over key-generation and the coins of SGN.

With regard to security we will consider the usual notion of unforgeability [GMR88] as well as strong unforgeability. Generate keys (pk, sk) via KG and run adversary A on input pk. It is allowed a chosen-message attack via access to $\mathsf{SGN}(sk, \cdot)$ oracle, and eventually outputs a pair M, S. Let $M_1, \ldots, M_s$ denote the messages A queries to its oracle, and $S_1, \ldots, S_s$ the responses returned. We make some simplifying assumptions, namely that when considering forgery $M \notin \{M_1, \ldots, M_s\}$ and when considering strong forgery $(M, S) \notin \{(M_1, S_1), \ldots, (M_s, S_s)\}$. Then, in either case, A wins if $\mathsf{VF}(pk, M, S) = 1$. We let $\mathbf{Adv}^{\mathrm{UF}}(A)$ and $\mathbf{Adv}^{\mathrm{SUF}}(A)$ denote, respectively, the probability that A wins in the case of standard and strong forgery. (The notation does not indicate the scheme, which will hopefully be clear from the context.) We denote the corresponding security notions by UF and SUF.

For pedagogic purposes it is useful to consider relaxed notions SE-UF and SE-SUF where SE stands for "selective." Here A must first (before it even sees the public key) output a message M

whose signature it will later attempt to forge. Then it gets the signing oracle to mount its chosen-message attack, at the end of which it returns a signature S, winning if $\mathsf{VF}(pk, M, S) = 1$. As before, if $M_1, \ldots, M_s$ denote the messages A queries to its oracle, and $S_1, \ldots, S_s$ the responses returned, then when considering forgery it is assumed that $M \notin \{M_1, \ldots, M_s\}$ and when considering strong forgery that $(M, S) \notin \{(M_1, S_1), \ldots, (M_s, S_s)\}$.

In the random oracle model [BR93], $\mathsf{SGN}, \mathsf{VF}$ and A can all make queries to the random oracle, and probabilities now involve the choices of the random oracle as well. We make the simplifying assumption that A never repeats a query to the random oracle.

2.2 The BSL signature scheme

The BSL scheme [BSL01] is one of the simplest applications of pairings and a good introduction to the use of pairings. Its merit is to yield very short signatures, only 160-bits under appropriate instantiations. We fix a pairing $\mathbf{e}$: $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. It is important that we work in the asymmetric setting to get signatures as short as we have just stated. We assume there is an efficiently computable isomorphism ψ: $\mathbb{G}_2 \to \mathbb{G}_1$. Note it will be used only by the algorithms in the proof of security, not by the scheme itself. Let p be the common order of the groups. Let H be a hash function with range $\mathbb{G}_1$. The scheme is as follows:

Keys:
$sk = x$ where $x \xleftarrow{\$} \mathbb{Z}_p$
$pk = (g_2, X)$ where $g_2 \xleftarrow{\$} \mathbb{G}_2^*$; $X \leftarrow g_2^x \in \mathbb{G}_2$

Signing:
$\mathsf{SGN}(sk, M) = H(M)^x \in \mathbb{G}_1$

Now notice that if $S = \mathsf{SGN}(sk, M)$ then $(g_2, X, H(M), S)$ is a dh-tuple. So

$$\mathbf{e}(H(M), X) \;=\; \mathbf{e}(H(M), g_2^x) \;=\; \mathbf{e}(H(M), g_2)^x \;=\; \mathbf{e}(H(M)^x, g_2) \;=\; \mathbf{e}(S, g_2)\ .$$

This leads to the following verification procedure, whose consistency we have just verified:

Verification:
$\mathsf{VF}(pk, M, S) = 1$ if $S \in \mathbb{G}_1$ and $\mathbf{e}(H(M), X) = \mathbf{e}(S, g_2)$, and 0 otherwise

Put another way, we are appealing to Proposition 1.6. But it is easier to see it directly.

As we said above, signatures in this scheme can be only 160 bits long, as opposed to 320 bits in schemes like Schnorr [Sch91] or El Gamal [El 85]. Signing needs just one exponentiation, essentially the same as in the other schemes just mentioned. However, verification in the BSL scheme needs two pairings, while verification in the other schemes needs only two exponentiations. So verification is significantly slower. The authors argue that there are setting where signature size is more important than verification time. I don't doubt it, but I don't think this is what really makes the scheme interesting. Rather, it is the ease with which the scheme lends itself to applications like threshold and multi signatures, and gives rise to aggregate signatures, as we will see later. For the moment though our concern is the security of the scheme.

Theorem 2.1 The BSL signature scheme is (S)UF if CDH is hard and H is a random oracle. Concretely, if A is an adversary against the BSL scheme that makes at most q hash queries and s sign queries then there is an adversary B against CDH such that

$$\mathbf{Adv}^{\mathrm{(S)UF}}(A) \;\le\; e(s+1) \cdot \mathbf{Adv}^{\mathrm{CDH}}(B)\ .$$

Furthermore the running time of B is that of A plus the time for $O(q)$ exponentiations. ▮

The strong unforgeability is a simple consequence of the unforgeability in this case because the scheme has unique signatures, meaning if a public key (g_2, X) and message M are fixed then there is exactly one value that the verifier would accept as a signature of M, namely, $H(M)^x$ where $x = \mathrm{DLog}_{g_2}(X)$. So we focus on showing UF.

We make some simplifying assumptions. Namely, if A asks sign query M or outputs M as the message in its forgery, then it must have previously asked hash query M. Recall we have already assumed that A does not ask for the signature of the message it outputs in its forgery and does not repeat queries to the hash oracle.

First cut. The proof is essentially the same as for FDH-RSA [BR96, Cor00]. We want to build adversary B that on input g_2, X, h, where $g_2 \xleftarrow{\$} \mathbb{G}_2^*$, $x \xleftarrow{\$} \mathbb{Z}_p$, $X \leftarrow g_2^x$ and $h \xleftarrow{\$} \mathbb{G}_1^*$, returns h^x. B will as usual run A and reply to its hash and sign queries. B would like to reply to hash queries in such a way that, if M is a message for which A later requests a signature, then $H(M) = \psi(g_2)^\beta$ for some β chosen by B, while if M is the message on which A forges, then $H(M) = h$. In this way, B can reply to sign query M be returning $\psi(X)^\beta = \psi(g_2)^{x\beta} = H(M)^x$, and, on the other hand, the forgery that A returns, if valid, is exactly the quantity h^x that B wants to compute. The only problem is that, at the time a message M is queried to the hash oracle, B needs to decide in which of these two ways it will reply, but it does not at this point know whether M will be the message in the forgery or will later be a sign query. There are a couple of (by now, standard) ways around this. One, as in [BR96], is to guess which of the messages queried to the hash oracle will be in the forgery. Another, which yields a better security reduction, is to use the biased coin-flip approach of [Cor00]. Following [BSL01], we will do the latter. Here is the description of B–

Adversary $B(g_2, X, h)$ // Wants to compute h^x where $X = g_2^x$
  $i \leftarrow 0$; $g_1 \leftarrow \psi(g_2)$; $X_1 \leftarrow \psi(X)$
  Run A on input public key (g_2, X), replying to its oracle queries as follows:

  On hash query M:

  $i \leftarrow i + 1$; $M_i \leftarrow M$; $x_i \xleftarrow{\$} \mathbb{Z}_p$; $\delta_i \xleftarrow{\delta} \{0,1\}$
  If $\delta_i = 1$ then $H[M] \leftarrow g_1^{x_i}$ else $H[M] \leftarrow h g_1^{x_i}$
  Return $H[M]$

  On sign query M:

  Let j be such that $M = M_j$
  If $\delta_j = 1$ then $S_j \leftarrow X_1^{x_i}$; return S_j
  Else abort

  Let M, S be the output of A
  If $S \notin \mathbb{G}_1$ or $\mathbf{e}(H[M], X) \neq \mathbf{e}(S, g_2)$ then abort
  Let j be such that $M = M_j$

If $\delta_j = 0$ then $w \leftarrow SX_1^{-x_j}$; return w
Else abort

Above, $0 \le \delta \le 1$ is a parameter whose value we will shortly determine, and the notation $c \xleftarrow{\delta}$ means that c is the result of a δ-biased coin, meaning 1 with probability δ and 0 with probability $1-\delta$.

Now let SIGOK be the event that $\delta_j = 1$ for every j such that M_j was a sign query. Let FORGE be the event that A successfully forges, meaning $S \in \mathbb{G}_1$ and $\mathbf{e}(H[M], X) = \mathbf{e}(S, g_2)$. Let INV be the event that $\delta_j = 0$ where j is such that M_j is the message in A's forgery. Letting $f(\delta) = \delta^s(1-\delta)$ (recall s is an upper bound on the number of sign queries made by A) the analysis could proceed as follows:

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{CDH}}(B) &\ge \Pr[\textsc{forge} \wedge \textsc{inv}] \\
&\ge \Pr[\textsc{forge} \wedge \textsc{inv} \wedge \textsc{sigok}] \\
&= \Pr[\textsc{inv} \mid \textsc{forge} \wedge \textsc{sigok}] \cdot \Pr[\textsc{forge} \mid \textsc{sigok}] \cdot \Pr[\textsc{sigok}] \\
&\ge (1-\delta) \cdot \mathbf{Adv}^{\mathrm{UF}}(A) \cdot \delta^s \qquad (2.1) \\
&= f(\delta) \cdot \mathbf{Adv}^{\mathrm{UF}}(A)\ .
\end{aligned}$$

The function f has derivative $f'(\delta) = \delta^{s-1}[(s+1)\delta - s]$, which is 0 for $\delta = s/(s+1)$. This is a maximum for the function, as one can tell by looking at the sign of f', and the value of f at this maximum is

$$f\left(\frac{s}{s+1}\right) \;=\; \left(1 - \frac{1}{s+1}\right)^s \cdot \frac{1}{s+1} \;\ge\; \frac{1}{e} \cdot \frac{1}{s+1}\ ,$$

which proves the claimed bound. It remains to justify the above inequalities, of which the non-numbered ones are obvious. Regarding Equation (2.1), the claims that

$$\Pr[\textsc{inv} \mid \textsc{forge} \wedge \textsc{sigok}] = 1-\delta \qquad \text{and} \qquad \Pr[\textsc{sigok}] = \delta^s$$

seem pretty believable. The first is true because, if M_j is the message in the forgery and FORGE is true, then A does not learn anything about δ_j, and this coins has probability $1-\delta$ of being 0. The second is true because the event SIGOK happens exactly when $\delta_j = 1$ for every j such that M_j was a sign query. Finally, we are claiming $\Pr[\textsc{forge} \mid \textsc{sigok}] = \mathbf{Adv}^{\mathrm{UF}}(A)$. This is true because as long as A's sign queries are correctly answered (meaning, B does not abort in answering sign queries) then its view is the same as in the the real game.

QUESTIONS. So are you convinced? I confess that I don't find the justification of Equation (2.1) entirely convincing. The conditional probability $\Pr[\textsc{forge} \mid \textsc{sigok}]$ is by definition the ratio of two probabilities, namely it equals $\Pr[\textsc{forge}]/\Pr[\textsc{sigok}]$. It is difficult for me to see why the statement that "as long as A's sign queries are correctly answered then its view is the same as in the the real game" tells us that this ratio equals a probability, namely $\mathbf{Adv}^{\mathrm{UF}}(A)$, defined via a different game. That is, the justification is using some intuitive interpretation of the conditional probability and omitting to relate it to the actual mathematical definition. And, for me, the gap is more than formal. After all, A does get information about the δ_j values, namely that the ones corresponding to any sign queries to which it has received answers are equal to 1. Can't it use this to behave differently from the way it behaves in the real game?

procedure Initialize G_0, $\boxed{G_1}$
$g_2 \xleftarrow{\$} \mathbb{G}_2^*$; $x \xleftarrow{\$} \mathbb{Z}_p$; $X \leftarrow g_2^x$
$h \xleftarrow{\$} \mathbb{G}_1^*$; $i \leftarrow 0$; $g_1 \leftarrow \psi(g_2)$; $X_1 \leftarrow \psi(X)$
Return (g_2, X)

On hash query M:
$i \leftarrow i + 1$; $M_i \leftarrow M$; $x_i \xleftarrow{\$} \mathbb{Z}_p$; $\delta_i \xleftarrow{\delta} \{0,1\}$
If $\delta_i = 1$ then $H[M] \leftarrow g_1^{x_i}$
Else $H[M] \leftarrow h g_1^{x_i}$
Return $H[M]$

On sign query M:
Let j be such that $M = M_j$
$S_j \leftarrow 1$
If $\delta_j = 1$ then $S_j \leftarrow X_1^{x_i}$
Else $\mathsf{bad} \leftarrow \mathsf{true}$ $\boxed{;\ S_j \leftarrow H[M]^x}$
Return S_j

procedure Finalize(M, S)
If $S \notin \mathbb{G}_1$ or $\mathbf{e}(H[M], X) \neq \mathbf{e}(S, g_2)$
  then return $\perp$
Let j be such that $M = M_j$
$w \leftarrow \perp$
If $\delta_j = 0$ then $w \leftarrow S X_1^{-x_j}$
Else $\mathsf{bad} \leftarrow \mathsf{true}$ $\boxed{;\ w \leftarrow h^x}$
Return w

procedure Initialize G_2
$g_2 \xleftarrow{\$} \mathbb{G}_2^*$; $x \xleftarrow{\$} \mathbb{Z}_p$; $X \leftarrow g_2^x$
$h \xleftarrow{\$} \mathbb{G}_1^*$; $i \leftarrow 0$
Return (g_2, X)

On hash query M:
$i \leftarrow i + 1$; $M_i \leftarrow M$; $x_i \xleftarrow{\$} \mathbb{Z}_p$; $\delta_i \xleftarrow{\delta} \{0,1\}$
$H[M] \xleftarrow{\$} \mathbb{G}_1$
Return $H[M]$

On sign query M:
Let j be such that $M = M_j$
If $\delta_j = 1$ then $S_j \leftarrow H[M]^x$
Else $\mathsf{bad} \leftarrow \mathsf{true}$; $S_j \leftarrow H[M]^x$
Return S_j

procedure Finalize(M, S)
If $S \notin \mathbb{G}_1$ or $\mathbf{e}(H[M], X) \neq \mathbf{e}(S, g_2)$
  then return $\perp$
Let j be such that $M = M_j$
If $\delta_j = 0$ then $w \leftarrow h^x$
Else $\mathsf{bad} \leftarrow \mathsf{true}$; $w \leftarrow h^x$
Return w

Figure 2.1: Games for the proof of Theorem 2.1. Game G_1 includes the boxed statements while G_0 does not.

Proofs in the literature usually just leave the reader to verify statements like this. They are viewed as "obvious." I don't think the claim above is wrong, and certainly am not saying there is a gap in the proof of [BSL01]. But it is not obvious to me why the claim is true. Furthermore, later (eg. for Water's scheme), we will see claims that are even more difficult to verify yet inherently analogous. So I'd like to spend some time right now trying to get a proof with better justification. Actually, we will not attempt to justify the above claim. Rather we will do the proof slightly differently, altering adversary B a bit. We will use code-based game-playing [BR06].

GAMES. Fig. 2.1 presents a total of three games, and Fig. 2.2 presents one more. To explain them, let us begin by recalling some of the framework of [BR06]. A game consists of an Initialize procedure, procedures that respond to adversary oracle queries (in our case, two such, one to respond to hash queries and the other to respond to sign queries) and a Finalize procedure. We will be executing A with each of these games. The execution of A with G_i is determined as follows.

procedure Initialize G_3
$g_2 \xleftarrow{\$} \mathbb{G}_2^*$; $x \xleftarrow{\$} \mathbb{Z}_p$; $X \leftarrow g_2^x$; $h \xleftarrow{\$} \mathbb{G}_1^*$; $i \leftarrow 0$; return (g_2, X)
On hash query M:
$H[M] \xleftarrow{\$} \mathbb{G}_1$; return $H[M]$
On sign query M:
$i \leftarrow i + 1$; $M_i \leftarrow M$; return $H[M]^x$
procedure Finalize(M, S)
If $S \notin \mathbb{G}_1$ or $\mathbf{e}(H[M], X) \neq \mathbf{e}(S, g_2)$ then return $\perp$
For $j = 1, \ldots, i$ do
  $\delta_i \xleftarrow{\delta} \{0,1\}$; If $\delta_i = 0$ then $\mathsf{bad} \leftarrow \mathsf{true}$
$\delta_{i+1} \xleftarrow{\delta} \{0,1\}$; If $\delta_{i+1} = 1$ then $\mathsf{bad} \leftarrow \mathsf{true}$
Return h^x

Figure 2.2: Final game G_3 for the proof of Theorem 2.1.

First, the Initialize procedure executes, and its output, which is a public key (g_2, X) as given by the Return statement, is passed as input to A. Now A executes, its hash and sign queries being answered by the procedures for this purpose associated to G_i. The output M, S of A becomes the input to the Finalize procedure of G_i. The output of the game is whatever is returned by the Finalize procedure. We let "$G_i^A \Rightarrow v$" denote the event that the output of Game G_i, when executed with A, is the value v. We adopt the convention that boolean variables like bad are automatically initialized to false and arrays like $H[\cdot]$ begin everywhere undefined.

A pair of games is said to be identical-until-bad if they differ only in statements that follow the setting of the flag bad to true. Thus games G_0, G_1 are identical-until-bad. The Fundamental Lemma of game-playing of [BR06] applies to such games. Here we will not use it in the form stated in that paper, but rather use the following variant:

Lemma 2.2 Let H_0, H_1 be identical-until-bad games and A an adversary. Let GOOD be the event that the flag bad is never set to true. Then

$$\Pr\left[H_0^A \Rightarrow v \wedge \text{GOOD}\right] \;=\; \Pr\left[H_1^A \Rightarrow v \wedge \text{GOOD}\right]$$

for all v. ∎

The proof of the Fundamental Lemma in [BR96] establishes this, but it is in any case not hard to see why it is true. If we run either game and A with a particular sequence of coin tosses for which GOOD is true, the game outputs will be identical because the games are identical-until-bad.

A GAME-PLAYING PROOF. We prove Theorem 2.1 using the games of Figures 2.1 and 2.2, and the above Lemma. We define adversary B as follows. On input g_2, X, h as before, where $X = g_2^x$, it sets $i \leftarrow 0$; $g_1 \leftarrow \psi(g_2)$; $X_1 \leftarrow \psi(X)$, and provides public key (g_2, X) to A. In running A, it answers the latters oracle queries exactly as indicated in game G_0. When A halts with some output M, S,

our adversary B runs the finalize procedure of game G_0 on this input, and finally outputs the value w returned by this procedure. Notice that B can do this because none of the procedures of G_0 make use of x. (Remember, the boxed statements are not present in G_0.) Also notice that our B is the same as the the one in our first-cut proof above except for one slight difference, namely that it does not abort when it can't answer a sign query, but instead returns a default value to A, and similarly does not abort if it can't profit from the forgery, but just outputs a default. We clarify that B is not calling the procedures of the game; it is just duplicating most of the game code. To make this clear, let us specify B in detail (we make a few optimizations, such as omitting setting bad, for it has no use here)–

Adversary $B(g_2, X, h)$ // Wants to compute h^x where $X = g_2^x$
$i \leftarrow 0$; $g_1 \leftarrow \psi(g_2)$; $X_1 \leftarrow \psi(X)$
Run A on input public key (g_2, X), replying to its oracle queries as follows:

On hash query M:

$i \leftarrow i + 1$; $M_i \leftarrow M$; $x_i \xleftarrow{\$} \mathbb{Z}_p$; $\delta_i \xleftarrow{\delta} \{0,1\}$
If $\delta_i = 1$ then $H[M] \leftarrow g_1^{x_i}$ else $H[M] \leftarrow h g_1^{x_i}$
Return $H[M]$

On sign query M:

Let j be such that $M = M_j$; $S_j \leftarrow 1$
If $\delta_j = 1$ then $S_j \leftarrow X_1^{x_i}$
Return S_j

Let M, S be the output of A
If $S \notin \mathbb{G}_1$ or $\mathbf{e}(H[M], X) \neq \mathbf{e}(S, g_2)$ then return $\perp$
Let j be such that $M = M_j$; $w \leftarrow \perp$
If $\delta_j = 0$ then $w \leftarrow S X_1^{-x_j}$
Return w

Now, for the analysis, let GOOD be the event that bad never gets set to true. We now claim a chain of inequalities (actually, mostly equalities). Notice that no conditional probabilities are involved. We will conclude assuming them, and then return to justify them:

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{CDH}}(B) &= \Pr\left[G_0^A \neq \perp\right] && (2.2)\\
&\geq \Pr\left[G_0^A \neq \perp \wedge \mathrm{GOOD}\right] \\
&= \Pr\left[G_1^A \neq \perp \wedge \mathrm{GOOD}\right] && (2.3)\\
&= \Pr\left[G_2^A \neq \perp \wedge \mathrm{GOOD}\right] && (2.4)\\
&= \Pr\left[G_3^A \neq \perp \wedge \mathrm{GOOD}\right] && (2.5)\\
&= \Pr\left[G_3^A \neq \perp\right] \cdot \Pr\left[\mathrm{GOOD}\right] && (2.6)\\
&= \mathbf{Adv}^{\mathrm{UF}}(A) \cdot \Pr\left[\mathrm{GOOD}\right] \ . && (2.7)
\end{aligned}$$

Assuming for the moment we believe these inequalities, the next task is to lower bound Pr[GOOD] in game G_3. But this is easy, because in that game, the relevant random choices are made at the very

end, and our previous calculation certainly applies. So what remains is to justify the inequalities.

Equation (2.2) is true by the way B was defined in terms of G_0 and taking into account that in the definition of $\mathbf{Adv}^{\mathrm{CDH}}(B)$ the quantities g_2, X, h are chosen just as by the initialize procedure of G_0. Games G_0, G_1 are identical-until-bad and hence Lemma 2.2 implies Equation (2.3). But now notice that in G_1, the reply to a hash oracle query M_i is a random element of $\mathbb{G}_1$ regardless of the value of δ_i. Similarly, the reply to a sign query M is always $H[M]^x$, even though it is computed in two different ways depending on the value of δ_j where $M = M_j$. And as long as $S \in \mathbb{G}_1$ and $\mathbf{e}(H[M], X) = \mathbf{e}(S, g_2)$, the output of the finalize procedure is h^x regardless of the value of δ_j. Game G_2 just makes the choices directly. We have justified Equation (2.4). In G_2, the choices of δ_i, and whether or not bad is set to true, do not affect the answers to queries or the output of the finalize procedure. Game G_3 delays these choices, making them and setting bad in its finalize procedure. We have justified Equation (2.5). Now it is easy to see that the events "$G_3^A \neq \perp$" and GOOD are independent in the execution of G_3 with A. This is because the choices determining the latter are made after the output of the game has been determined. Thus we have justified Equation (2.6). Indeed the purpose of the game-chain was exactly to come to a point where this claim of independence and this step was evident. Finally we observe that game G_3 answers oracle queries exactly as the game defining the UF-CMA security of A, justifying Equation (2.7).

So, what do you think? Is it believable now? Some people would say we are belaboring an obvious claim. Others would say the game-based proof is not clearer than the first-cut one. You can form your own opinions. As Shimon Even has been quoted as saying, "a proof is whatever convinces me."

Games are not a panacea. One can make errors in game-based reasoning too, and one can also create game-based proofs that are hard to verify, typically by claiming some relation between games that is not obvious. It is hard to know exactly what "obvious" means in any context, but obviously it means different things to different people.

I now present a problem. A problem is intended to be more challenging than an exercise, possibly open, open-ended or not very precisely defined. But it might have research potential.

Problem 2.3 It seems to me there is something very general underlying what goes on here. Can you formulate some sort of general lemma about games that can be immediately applied here and elsewhere? ▌

A SECURITY IMPROVEMENT.

Bibliography

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS 93*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, Saragossa, Spain, May 12–16, 1996. Springer-Verlag, Berlin, Germany.

[BR06] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, St. Petersburg, Russia, May 28 – June 1, 2006. Springer-Verlag, Berlin, Germany. `http://eprint.iacr.org/2004/331`.

[BSL01] Dan Boneh, Hovav Shacham, and Ben Lynn. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer-Verlag, Berlin, Germany.

[Cor00] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 229–235, Santa Barbara, CA, USA, August 20–24, 2000. Springer-Verlag, Berlin, Germany.

[El 85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 10–18, Santa Barbara, CA, USA, August 19–23, 1985. Springer-Verlag, Berlin, Germany.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.