J. Cryptology (2007) 20: 493–514
DOI: 10.1007/s00145-007-0549-3

Journal of
CRYPTOLOGY

Efficient Signature Schemes with Tight Reductions to the Diffie–Hellman Problems*

Eu-Jin Goh
Computer Science Department, Stanford University,
Stanford, CA 94305, U.S.A.
eujin@cs.stanford.edu

Stanisław Jarecki
School of Information and Computer Science,
University of California at Irvine,
Irvine, CA 92697, U.S.A.
stasio@ics.uci.edu

Jonathan Katz and Nan Wang
Department of Computer Science, University of Maryland,
College Park, MD 20742, U.S.A.
{jkatz,nwang}@cs.umd.edu

Communicated by Matthew Franklin

Received 9 October 2005 and revised 29 September 2006
Online publication 13 July 2007

Abstract. We propose and analyze two efficient signature schemes whose security is tightly related to the Diffie–Hellman problems in the random oracle model. The security of our first scheme relies on the hardness of the *computational* Diffie–Hellman problem; the security of our second scheme—which is more efficient than the first—is based on the hardness of the *decisional* Diffie–Hellman problem, a stronger assumption.

Given the current state of the art, it is as difficult to solve the Diffie–Hellman problems as it is to solve the discrete logarithm problem in many groups of cryptographic interest. Thus, the signature schemes shown here can currently offer substantially better efficiency (for a given level of provable security) than existing schemes based on the discrete logarithm assumption.

The techniques we introduce can also be applied in a wide variety of settings to yield more efficient cryptographic schemes (based on various number-theoretic assumptions) with tight security reductions.

Key words. Signature schemes, Diffie–Hellman assumptions.

* This paper combines results that appeared in "A Signature Scheme as Secure as the Diffie-Hellman Problem" [23] presented at Eurocrypt 2003 and "Efficiency Improvements for Signature Schemes with Tight Security Reductions" [26] presented at ACM CCCS 2003. The work by Stanisław Jarecki was done while at Stanford University. Jonathan Katz and Nan Wang were supported by NSF Grants #0310751 and #0208005.

1. Introduction

One focus of modern cryptography has been the construction of signature schemes that can be rigorously proven secure based on specific computational assumptions. A proof of security for a given construction generally proceeds by demonstrating a *reduction* which shows how a polynomial-time adversary "breaking" the signature scheme can be used to solve in polynomial time some underlying problem assumed to be difficult (e.g., inverting a one-way function); it follows that if the underlying problem is *truly* difficult for all polynomial-time algorithms, then the given signature scheme is indeed secure. Classically, such results have been *asymptotic*; namely, the security reduction only demonstrates that no polynomial-time adversary can forge a signature with non-negligible probability, where both the running time of the adversary and its probability of forgery are measured as a function of some security parameter k. As first emphasized by Bellare and Rogaway [4], however, such results say nothing about the security of a given scheme in *practice* for a particular choice of security parameter, and against adversaries investing a particular amount of computational effort. Consequently, asymptotic security reductions by themselves do not enable practically meaningful efficiency comparisons between two signatures schemes: the efficiency of two schemes can be meaningfully compared only when they achieve the same level of security, yet asymptotic security results do not by themselves provide enough information to determine when that is the case.

A simplified example illustrates the point. Assume we are given a discrete logarithm-based signature scheme along with a security proof guaranteeing that (under current assumptions regarding the hardness of the discrete logarithm problem) any adversary expending 1 year of computational effort can "break" the scheme with probability at most $a \cdot 2^{-b \cdot k}$, where $a, b > 0$ are constants and k is (say) the bit-length of the order of the group used in the scheme. In an asymptotic sense, this scheme is secure regardless of a, b. In practice, however, we do not know what value of k to choose to achieve some desired level of security unless a and b are known. For this reason, the values of a and b are crucial for determining the actual efficiency of the scheme. For example, for a desired security level (i.e., probability of forgery) of 2^{-32} against adversaries investing 1 year of effort, having $a \approx 1$ and $b \approx \frac{1}{10}$ means that we should set $k \approx 320$. On the other hand, if $a \approx 2^{32}$ and $b \approx \frac{1}{20}$, then we require $k \approx 1280$, which implies a concomitant decrease in efficiency to achieve the same level of security.

This motivates an emphasis on *concrete* security reductions that give explicit bounds on the adversary's success probability (i.e., its probability of forging a signature) as a function of its expended resources [4], [15], [29], [16]. It also illustrates the importance of designing schemes with *tight* security reductions: that is, reductions showing that the success probability of an adversary running in some time t is roughly *equal* to the probability of solving the underlying hard problem in roughly the same amount of time. At least intuitively, a tight reduction is the best one can hope for, as any scheme based on (a single instance of) any particular "hard" problem seemingly cannot be more difficult to break than the problem itself is to solve.

1.1. *Previous Work*

The above considerations have sparked a significant amount of research aimed at finding efficient signature schemes with tight security reductions. Though there exist signature

schemes (e.g., [24] and [17]) with tight security reductions in the so-called standard model, these schemes are generally considered too inefficient for practical use and so recent attention has turned to schemes analyzed in the random oracle model.[1] We first describe progress in this regard for schemes based on trapdoor permutations (with RSA serving as a specific example), and then discuss schemes based on the discrete logarithm problem.

Signature schemes based on trapdoor permutations. For some fixed value of the security parameter, let ε' be an assumed upper bound on the probability of inverting a given trapdoor permutation in some time t'. The full domain hash (FDH) signature scheme [3], [4] bounds the success probability of any adversary running in time $t \approx t'$ by $\varepsilon \approx (q_s + q_h)\varepsilon'$, where q_s is the number of signatures the adversary obtains from the legitimate signer, and q_h represents the number of times the adversary evaluates the hash function (formally, q_h is the number of queries the adversary makes to the random oracle).[2] Subsequently, Coron [15] showed how to achieve the better security reduction $\varepsilon \approx q_s\varepsilon'$ for FDH if the underlying trapdoor permutation is random self-reducible as is the case for, e.g., RSA. Dodis and Reyzin [19], generalizing Coron's work, show that a similar result holds for any trapdoor permutation induced by a family of claw-free permutations.

The probabilistic signature scheme (PSS) [4] was introduced precisely to obtain a tight security reduction for the specific case when RSA is used as the underlying trapdoor permutation (although it was later shown that the tight reduction holds also for more general classes of trapdoor permutations [16], [19]). PSS uses a randomly chosen string (a "salt") each time a message is signed and obtains the tight security reduction $\varepsilon \approx \varepsilon'$. Coron subsequently observed [16] that the length of the salt can be significantly reduced while obtaining essentially the same security bound.

The above signature schemes are essentially all based on the classical "hash-and-sign" paradigm. An alternate approach is to use the Fiat–Shamir methodology [20] for converting 3-round, public-coin identification schemes to signature schemes. Applying this approach to some specific identification schemes yields signature schemes based on a number of specific trapdoor permutations, including RSA. Unfortunately, the best known security reduction for schemes constructed using the Fiat–Shamir transformation relies on the "forking lemma" of Pointcheval and Stern [30], with some improvements in the analysis due to Micali and Reyzin [29]. Applying this lemma results in a very loose security reduction: roughly speaking, given an adversary running in time t and "breaking" a signature scheme with probability ε, the reduction yields an algorithm solving the underlying hard problem with constant probability in time $t' \approx \Theta(q_h t \varepsilon^{-1})$ (where the constant term in the big-Θ notation is greater than 1). Substituting $q_h \approx 2^{60}$, $\varepsilon \approx 2^{-60}$ gives a weak result unless the time t' that is assumed to be required to solve the underlying hard problem is huge.

[1] The random oracle model [20], [3] assumes a public, random function which is accessible by all parties. In practice, this oracle is instantiated by a cryptographic hash function. Although security can no longer be guaranteed for any such instantiation [10], a proof in the random oracle model does seem to indicate that there are no "inherent" weaknesses in the scheme and, in practice, serves as a useful validation tool for cryptographic constructions.

[2] Typical suggested values for these parameters are $q_s \approx 2^{30}$ and $q_h \approx 2^{60}$ [4], [15], [16].

Micali and Reyzin [29] show a modification of the Fiat–Shamir transformation that leads to tighter security reductions, but applies only to specific identification schemes. Using their transformation, Micali and Reyzin show signature schemes with tight security reductions based on some specific trapdoor permutations, including RSA. A recent result of Fischlin [21] shows an alternate way of modifying the Fiat–Shamir transformation so as to obtain a tight security reduction; the schemes resulting from this approach, however, are relatively inefficient.

Signature schemes based on the discrete logarithm problem. In contrast to the case of trapdoor permutations, there has been significantly less progress designing signature schemes with tight security reductions to the hardness of computing discrete logarithms (or related problems). DSS [33], perhaps the most widely used discrete logarithm-based scheme, has no known proof of security. Existing provably secure schemes based on the discrete logarithm assumption, such as those by Schnorr [31], an El Gamal [22] variant suggested by Pointcheval and Stern [30], and a DSS variant by Brickell et al. [8], rely on (variants of) the forking lemma for their proofs of security and therefore have very loose security reductions.[3] (The work of Micali and Reyzin, mentioned earlier, cannot be applied to any of these schemes.)

1.2. *Our Contributions*

We design two efficient signature schemes with *tight* security reductions (in the random oracle model) to problems related to the hardness of computing discrete logarithms. Our first scheme relies on the *computational* Diffie–Hellman (CDH) [18] problem and is based on a scheme previously suggested—but not proven secure—by Chaum et al. [11], [13]. Our second scheme is more efficient, but its security is based on the stronger *decisional* Diffie–Hellman (DDH) assumption. See Section 2.2 for formal definitions of these two assumptions.

Although both Diffie–Hellman assumptions are, technically speaking, stronger than the discrete logarithm assumption, for a variety of well-studied cryptographic groups it is currently not known how to solve the Diffie–Hellman problems any faster than what can be achieved by solving the discrete logarithm problem itself [5], [27]. Moreover, there is some theoretical evidence that in certain groups the computational Diffie–Hellman assumption may be equivalent to the discrete logarithm assumption [32], [5], [27]. For such groups, then, our schemes offer a marked improvement compared with previous signature schemes based on the discrete logarithm problem.

We compare the efficiency of our schemes in Table 1, focusing on the number of group exponentiations they each require (other operations are ignored as they are dominated by these costs). In the table, the computational cost of a multi-exponentiation (that is, computing $g^a h^b$) is assumed to be equivalent to 1.5 exponentiations [28, Section 14.6.1(iii)]. "Off-line" computation refers to computation that may be performed before the message to be signed is known, while "on-line" computation must be done after this point. The

[3] For the Schnorr signature scheme, a tight reduction is known in the *generic group model* [32]. This model considers only algorithms which are oblivious to the representation of group elements. For certain groups, however, there are known algorithms (e.g., the index-calculus method) that do not fall in this category.

Table 1. Efficiency of our schemes in a cyclic group $\mathbb{G}$ of prime order q.[a]

	Signing				Public key	Signature	
	Off-line	On-line	Total	Verifying	length	length	Assumption
Scheme 1	$(1^*, 0)$	$(2, 1)$	$(3, 1)$	$(3, 1)$	G	$G + \|q\| + k + 1$	CDH
Scheme 2	2^*	0	2^*	3^*	$3G$	$\|q\| + k$	DDH

[a] We assume G bits are used to represent elements of $\mathbb{G}$ (note that $G \geq |q| \stackrel{\text{def}}{=} \lceil \log_2 q \rceil$), and $k < |q|$ is a parameter that affects the tightness of the security reduction. The computational cost for Scheme 1 is denoted by (a, b), where a is the number of exponentiations in $\mathbb{G}$, and b is the number of "hash-to-$\mathbb{G}$" operations. The computational cost for Scheme 2 reflects the number of exponentiations in $\mathbb{G}$. A "*" indicates exponentiations with respect to a fixed base, where pre-computation can be used to improve efficiency [28, Section 14.6.3]. See the text for additional discussion.

tabulated values represent the efficiency of our schemes as described in Sections 3 and 4; however, various trade-offs are possible and these are not reflected in the table.

The parameter k affects the tightness of the security reduction. Roughly speaking, k should be set such that a probability of forgery $\approx q_h \cdot 2^{-k}$ is considered acceptable.

For our first scheme, we make some mild technical assumptions on the underlying group $\mathbb{G}$ that are discussed in further detail in Section 2.3. These technical assumptions can be avoided altogether at the expense of performing a group membership test during signature verification (which, depending on $\mathbb{G}$, may impose noticeable additional computation). Our first scheme also requires a random oracle mapping its inputs to elements of $\mathbb{G}$. Depending on the specific group being used, this "hash-to-$\mathbb{G}$" operation may also introduce noticeable computational cost. (See Section 2.4 for some discussion on this point.) Rather than assume any particular choice of $\mathbb{G}$, we have explicitly tabulated such operations for our first scheme.

Other applications. The techniques used in constructing our first scheme can be profitably applied in other contexts to yield efficiency improvements along with tight security reductions. For example, they can be used to obtain a tight proof of security while avoiding the need for a random salt in the PSS and PSS-R signature schemes [4], [16] as well as the short signature scheme of Boneh et al. [7]; they can also be used to improve the security reduction for the Boneh–Franklin identity-based encryption scheme [6]. The ideas used in constructing our second scheme can be applied to yield other signature schemes with tight security reductions to *decisional* problems, rather than loose security reductions to computational problems. These applications are discussed further in Section 1.3 and [26].

1.3. *Overview of Our Techniques*

We now give a high-level description of the main ideas underlying our two constructions; as preparation, we first review some relevant background.

Proving equality of discrete logarithms. Let $\mathbb{G}$ be a group of prime order q. We begin by reviewing the standard protocol for proving equality of discrete logarithms [12], [9], which is based on Schnorr's proof of knowledge of a discrete logarithm [31]. In the protocol a prover has values $g, h, y_1, y_2 \in \mathbb{G}$, with $g, h \neq 1$, together with an exponent

$x \in \mathbb{Z}_q$ such that $g^x = y_1$ and $h^x = y_2$. To prove to a verifier (who also knows g, h, y_1, y_2) that $\log_g y_1 = \log_h y_2$, the two parties execute the following interactive protocol:

1. The prover chooses random $r \in \mathbb{Z}_q$ and sends $A = g^r,\ B = h^r$ to the verifier.
2. The verifier sends to the prover a random challenge $c \in \{0,1\}^k$, where $2^k \leq q$ and c is interpreted as an integer in $\{0, \ldots, q-1\}$.
3. The prover replies with $s = cx + r \bmod q$.
4. The verifier accepts if and only if $A \stackrel{?}{=} g^s y_1^{-c}$ and $B \stackrel{?}{=} h^s y_2^{-c}$.

It is well known that the above protocol is *honest-verifier zero-knowledge*: a simulator given (g, h, y_1, y_2) can choose $c \leftarrow \{0,1\}^k$ and $s \leftarrow \mathbb{Z}_q$, and then set $A = g^s y_1^{-c}$ and $B = h^s y_2^{-c}$, and it can be verified that the resulting transcript (A, B, c, s) is distributed identically to the transcript of an execution of the protocol between a prover (who knows x) and an honest verifier.

It is also easy to verify that the above protocol is *sound*; in particular, if $\log_g y_1 \neq \log_h y_2$, then for any A, B sent by a cheating prover there is at most one value c for which the prover can respond correctly. This assumes that $y_2 \in \mathbb{G}$; in Section 2.3 we show a generalization of this result for the case when y_2 may not be in $\mathbb{G}$. (We also generalize to the case when g or h may be equal to 1.)

Using the Fiat–Shamir transformation [20], the above protocol can be made non-interactive using a hash function H modeled as a random oracle. Here, the prover computes A and B as above, sets $c = H(g, h, y_1, y_2, A, B)$, and then computes s as before; it then sends the proof $\pi \stackrel{\text{def}}{=} (c, s)$ to the verifier. The verifier computes $A' = g^s y_1^{-c}$ and $B' = h^s y_2^{-c}$ and accepts if and only if $c \stackrel{?}{=} H(g, h, y_1, y_2, A', B')$.

The Schnorr signature scheme. The protocol described previously can be adapted easily to give an honest-verifier zero-knowledge proof of knowledge of a discrete logarithm (see [31]). By making this protocol non-interactive using the Fiat–Shamir transformation, we obtain the Schnorr signature scheme [31]. In this scheme the signer's public key is (g, y_1) and its secret key is $x = \log_g y_1$; a signature on a message m is a non-interactive proof of knowledge of $\log_g y_1$ (we omit some details). In the proof of security [31], [30], [29], an adversary attacking the scheme is "rewound" in order to "extract" the value $\log_g y_1$; this rewinding, however, results in a poor security reduction (via the forking lemma) as discussed earlier in the Introduction.

A signature scheme based on the CDH problem.[4] As we have just remarked, a drawback of the Schnorr scheme is that the security reduction relies on the *proof of knowledge* property of an interactive proof system; this seems inherently to yield a loose security reduction. In contrast, we show that it is possible to design schemes based on the CDH assumption whose security relies only on the fact that the proof system is *sound* (as well as honest-verifier zero knowledge). This avoids the need for rewinding, and hence results in an improved security reduction.

[4] Informally, the CDH problem in $\mathbb{G}$ is: given a random tuple (g, h, y_1), output y_2 such that $\log_h y_2 = \log_g y_1$. The CDH assumption for $\mathbb{G}$ is that the CDH problem in $\mathbb{G}$ is "hard" to solve. See Section 2.2 for a formal definition.

We first recall a scheme suggested (but not analyzed) by Chaum and Pedersen [13]: The signer's public key is $(g, y_1) \in \mathbb{G}^2$ and its secret key is $x = \log_g y_1$. Let H' be a random oracle (independent of H) whose range is $\mathbb{G}$. To sign message m, the signer computes $h = H'(m)$ and $y_2 = h^x$; the signer then generates a proof π showing that $\log_g y_1 = \log_h y_2$ using the protocol for proving the equality of discrete logarithms described earlier. The signature (y_2, π) is verified in the natural way.

Security of the Chaum–Pedersen scheme was not previously analyzed, perhaps because it was viewed as a variant of Schnorr signatures. Our initial work [23] noted that a non-tight security reduction for the Chaum–Pedersen scheme can be derived by following the original analysis of FDH [3]: if ε' is an upper bound on the probability that the CDH problem can be solved in time t', then the success probability of any adversary attacking the Chaum–Pedersen scheme and running in time $t \approx t'$ is at most $\varepsilon \approx q_h \varepsilon'$. Sketching the proof, a simulator given an instance (g, h, y_1) of the CDH problem (where the simulator's goal is to output y_2 satisfying $\log_g y_1 = \log_h y_2$) sets the public key to (g, y_1). Next, it guesses an index i for the hash query to H' that the adversary will use in its forgery and sets the output of the ith hash query $H'(m_i)$ to be h. The simulator sets the output of all other hash queries $H'(m_j)$ to be g^{α_j} for random and independent $\alpha_j \in \mathbb{Z}_q$. If the simulator is asked to sign a message $m_j \neq m_i$, it can compute the (correct) value $y_1^{\alpha_j}$ and then simulate a proof of equality π. Furthermore, if the adversary does indeed forge a signature on m_i, then the simulator can (with overwhelming probability) recover from this forgery the desired solution to the original instance of the CDH problem.

The reduction sketched above results in a loss of a factor q_h in the security reduction since the simulator must guess the correct index of the hash query that the adversary uses to create the forgery. This can be improved using the approach of Coron [15]. Here, for all i the simulator answers the ith hash query $H'(m_i)$ by returning g^{α_i} with probability ρ, but returning $h \cdot g^{\alpha_i}$ with probability $1 - \rho$ (again, the $\{\alpha_i\}$ are independent and uniformly distributed in $\mathbb{Z}_q$). Say m_i is a message of the *first type* if the simulator responded with g^{α_i}, and is of the *second type* otherwise. The key observation is that the simulator can correctly answer any signing queries for messages of the first type, and can (with all but negligible probability) compute the solution to the given instance of the CDH problem whenever the adversary's forgery is on a message of the second type. By choosing ρ appropriately, one can obtain the improved security reduction $\varepsilon \approx q_s \varepsilon'$.

Our initial work [23] also showed a variant of the Chaum–Pedersen scheme (building on the ideas of [4] and [16]) which has a *tight* security reduction to the CDH problem. In this scheme—called the EDL scheme in [23]—the public and secret keys are as above. When signing a message, the signer first chooses a random "salt" $r \in \{0, 1\}^k$, computes $h = H'(r, m)$, and then proceeds as above (the signature now includes r as well). In the proof of security for this scheme, the simulator answers *all* of the adversary's hash queries to H' with $h \cdot g^{\alpha_i}$. To respond to a signing query for a message m, the simulator chooses $r \in \{0, 1\}^k$ and checks if the adversary had previously queried $H'(r, m)$. If so, the simulator aborts; otherwise, the simulator sets $H'(r, m) = g^{\alpha_j}$ and proceeds with the simulation as before. Now any successful forgery by the adversary allows the simulator, with overwhelming probability, to compute a solution to the original instance of the CDH problem. Furthermore, by setting k appropriately, it is possible to ensure that the simulator does not abort "too often." A drawback of this modified scheme is that signature length is increased by k, the length of the salt.

The reader is referred to [23] for full descriptions and proofs of the schemes sketched in the preceding paragraphs; we do not include them here because the scheme described next improves on the above in all important respects.

Scheme 1: *An improved scheme based on the CDH problem.* The first scheme presented in this paper improves on the EDL scheme in that security is still tightly related to the CDH problem, but the signature is shorter and a random salt is no longer needed. In this improved scheme the public and secret keys are as in the EDL scheme. The main difference is that the signer now has a hidden, random "selector bit" b_m associated with each message m that it signs; we defer for now the details of how this selector bit is determined. To sign message m, the signer computes $h = H'(b_m, m)$ and then proceeds as in the EDL scheme but includes b_m in the signature. Verification is done in the natural way. In the proof of security we now have the simulator (who is again given a random instance (g, h, y_1) of the CDH problem) proceed as follows: it answers the hash query $H'(b_m, m)$ with g^{α_m} but answers the hash query $H'(\bar{b}_m, m)$ with $h \cdot g^{\alpha_m}$. Note that the simulator can answer *all* of the adversary's requests to sign any given message m since the simulator knows $\log_g H(b_m, m)$; on the other hand, any forgery by the adversary allows the simulator to solve the original instance of the CDH problem with probability essentially $1/2$ since the adversary does not know b_m for any message m that has not been signed. Hence, we obtain a tight security reduction to the CDH problem. Full details are given in the proof of Theorem 1.

Scheme 2: *A more efficient scheme based on the DDH problem.*[5] All the schemes based on the CDH assumption outlined above follow the same basic paradigm: the message m determines a value $h \in \mathbb{G}$; the signer computes h^x and then proves that this was done correctly. We observe that if one is willing to base security on the stronger DDH assumption, then *it is unnecessary to use a new h for each message signed.* In particular, consider the signature scheme where the public key is (g, h, y_1, y_2) and the secret key is a value x such that $x = \log_g y_1 = \log_h y_2$. Now, a signature is simply a proof that $\log_g y_1 = \log_h y_2$ with one subtlety: to "bind" the proof to a particular message m we include m as one of the inputs to the hash function H (recall that H is the hash function used to implement the Fiat–Shamir transformation). For the proof of security, given an adversary that succeeds in attacking the scheme with probability ε, consider the simulator which is given as input a tuple (g, h, y_1, y_2). The simulator sets this value as its public key. When the adversary requests a signature, the simulator provides this signature by simulating the proof of equality of discrete logarithms for the tuple contained in the public key. Hash queries by the adversary are answered by returning a random value. If $\log_g y_1 = \log_h y_2$ then the adversary's view is essentially identical to its view when attacking a "real" instance of the signature scheme, and so it succeeds in forging a signature with probability roughly ε. On the other hand, if $\log_g y_1 \neq \log_h y_2$ then soundness of the proof system implies that the adversary succeeds in forging a signature

[5] Informally, the DDH problem in $\mathbb{G}$ is: given a tuple (g, h, y_1, y_2), decide whether $\log_g y_1 = \log_h y_2$. The DDH assumption for $\mathbb{G}$ is that it is "hard" to solve the DDH problem in $\mathbb{G}$ with non-negligible advantage, where the advantage is related to the probability of deciding correctly minus $1/2$. See Section 2.2 for a formal definition.

with only negligible probability. This simulator can thus be used to solve the DDH problem with advantage roughly ε, meaning that we get a tight security reduction. The proof of Theorem 2 gives further details.

Other applications of our techniques. The idea of using a "selector bit" to obtain a tight proof of security (as we do in our first scheme) can be applied in other contexts to avoid using a long(er) salt. For example, it immediately applies to the BLS signature scheme [7], as well as to the PSS and PSS-R signature schemes [4], [16] when based on the RSA trapdoor permutation. In fact, in the *random permutation model*[6] this technique can be applied to obtain a signature scheme supporting message recovery which has a tight security reduction and *optimal* signature length. Finally, the technique can also be used to improve the security reduction in the Boneh–Franklin identity-based encryption scheme [6]. These applications are discussed in our previous work [26].

The idea (as used in our second scheme) of using the Fiat–Shamir transformation to provide a *proof* rather than a *proof of knowledge* can be used more generally to achieve a tight security reduction based on a *decisional* problem rather than a non-tight security reduction based on a *computational* problem. For example, in the Fiat–Shamir signature scheme [20] the signer includes in his public key quadratic residues $\{y_i\}$ (modulo a composite N), and signs a message by proving knowledge of the square roots of these values. Using the forking lemma of Pointcheval and Stern [30], one obtains a non-tight security reduction to the hardness of computing square roots modulo N. On the other hand, by having the signer prove that the $\{y_i\}$ are all quadratic residues (without necessarily proving *knowledge* of their square roots), one obtains a *tight* security reduction to the hardness of deciding quadratic residuosity modulo N.

1.4. *Subsequent Work*

In work building on our own, Chevallier-Mames [14] shows a signature scheme whose efficiency and security are roughly equivalent to our Scheme 1 with the exception that all exponentiations during signing can be done *off-line*.

2. Definitions and Preliminaries

We review the standard definitions for signature schemes, as well as the computational and decisional Diffie–Hellman assumptions. We also more formally consider the soundness property of the protocol for proving equality of discrete logarithms that was briefly described in Section 1.3, and discuss how to implement a random oracle mapping to certain *groups* given as a building block a random oracle mapping to *bit strings*.

2.1. *Signature Schemes*

We provide both a functional definition and a security definition of signature schemes. Since we analyze our schemes in terms of their concrete security, our definitions are

[6] The random permutation model (see [26]) assumes a public, random, invertible permutation which is accessible by all parties. In practice, this would be instantiated using a block cipher with a fixed key.

concrete rather than asymptotic and do not explicitly refer to any security parameter. (Our results, however, imply security in the asymptotic sense as well.) Our definitions of concrete security assume a fixed computational model: e.g., Turing machines with binary alphabet and an upper bound on the number of states.

Since our schemes are analyzed in the random oracle model, we explicitly incorporate a random oracle H in our definitions. We note that multiple, independent random oracles can be derived from a single random oracle in a straightforward[7] way; thus, it suffices to consider only the case of a single random oracle. We let Ω denote the space from which H is selected; namely, the set of all functions defined over the appropriate domain and range.

Definition 2.1. A signature scheme is a tuple of probabilistic algorithms (Gen, Sign, Vrfy) over a message space $\mathcal{M}$ such that:

- The key generation algorithm Gen outputs a public key PK and a secret key SK.
- The signing algorithm $\mathsf{Sign}^{H(\cdot)}$ takes a secret key SK and a message $m \in \mathcal{M}$ as inputs and returns a signature σ.
- The verification algorithm $\mathsf{Vrfy}^{H(\cdot)}$ takes a public key PK, a message $m \in \mathcal{M}$, and a signature σ as inputs and returns accept or reject.

We make the standard correctness requirement: for all $H \in \Omega$, all (SK, PK) output by Gen, and all $m \in \mathcal{M}$, we have $\mathsf{Vrfy}^{H(\cdot)}_{PK}(m, \mathsf{Sign}^{H(\cdot)}_{SK}(m)) = \mathsf{accept}$.

For simplicity, in the rest of the paper we omit the explicit dependence of the signing and verification algorithms on H.

We now give the standard definition of existential unforgeability under adaptive chosen-message attacks [24]. We also define the notion of *strong* unforgeability; informally, this means that the adversary cannot even generate a new signature for a previously signed message.

Definition 2.2. Let $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme. An adversarial forging algorithm $\mathcal{F}$ $(t, q_h, q_s, \varepsilon)$-breaks this scheme if $\mathcal{F}$ runs in time at most t, makes at most q_h hash queries (that is, queries to the random oracle H) and at most q_s signing queries, and furthermore

$$\Pr\left[\begin{matrix}(PK, SK) \leftarrow \mathsf{Gen};\ H \leftarrow \Omega; \\ (m, \sigma) \leftarrow \mathcal{F}^{\mathsf{Sign}_{SK}(\cdot), H(\cdot)}(PK)\end{matrix} : m \notin \mathcal{M}^* \bigwedge \mathsf{Vrfy}_{PK}(m, \sigma) = \mathsf{accept}\right] \geq \varepsilon,$$

where $\mathcal{M}^*$ is the set of messages that $\mathcal{F}$ submitted to its signing oracle. In addition, we say that $\mathcal{F}$ $(t, q_h, q_s, \varepsilon)$-breaks this scheme in the strong sense if

$$\Pr\left[\begin{matrix}(PK, SK) \leftarrow \mathsf{Gen};\ H \leftarrow \Omega; \\ (m, \sigma) \leftarrow \mathcal{F}^{\mathsf{Sign}_{SK}(\cdot), H(\cdot)}(PK)\end{matrix} : (m, \sigma) \notin \Sigma^* \bigwedge \mathsf{Vrfy}_{PK}(m, \sigma) = \mathsf{accept}\right] \geq \varepsilon,$$

where Σ^* is the set of pairs (m, σ) such that σ was the response to an adversarial query $\mathsf{Sign}_{SK}(m)$.

[7] For example, given a random oracle H we may construct random oracles H_0, H_1 that are independent of each other by defining $H_0(x) = H(0x)$ and $H_1(x) = H(1x)$.

We say that signature scheme (Gen, Sign, Vrfy) is $(t, q_h, q_s, \varepsilon)$-secure in the sense of unforgeability (resp., strong unforgeability) if no forger can $(t, q_h, q_s, \varepsilon)$-break it (in the appropriate sense).

2.2. *The Diffie–Hellman Problems*

Let $\mathbb{G}$ be a finite, cyclic group of prime order q in which the group operation is represented multiplicatively, and let g be a fixed generator of $\mathbb{G}$. Given g and group elements g^x, g^y, the *computational* Diffie–Hellman (CDH) problem is to find the group element g^{xy}. (Equivalently, given h, $y_1 \in \mathbb{G}$ the problem is to compute $h^{\log_g y_1}$.) Informally, the CDH problem is "hard" in $\mathbb{G}$ if no efficient algorithm can solve the CDH problem with high probability for random g^x, g^y. The following definition makes this more concrete:

Definition 2.3. An algorithm A is said to (t, ε)-solve the CDH problem in $\mathbb{G}$ if A runs in time at most t and furthermore

$$\Pr[x, y \leftarrow \mathbb{Z}_q\colon\ A(g, g^x, g^y) = g^{xy}] \geq \varepsilon.$$

We say that $\mathbb{G}$ is a (t, ε)-CDH group if no algorithm (t, ε)-solves the CDH problem in $\mathbb{G}$.

The *decisional* Diffie–Hellman (DDH) problem may be described, informally, as the problem of distinguishing between tuples of the form (g, g^x, g^y, g^{xy}) for random $x, y \in \mathbb{Z}_q$ (these are called "Diffie–Hellman tuples") and tuples of the form (g, g^x, g^y, g^z) for random $x, y, z \in \mathbb{Z}_q$ (these are called "random tuples"). (Equivalently, the problem is to distinguish between tuples of the form $(g, h, y_1, h^{\log_g y_1})$ and tuples of the form (g, h, y_1, y_2) with y_2 uniformly distributed in $\mathbb{G}$.) The DDH problem is "hard" in $\mathbb{G}$ if no efficient algorithm can distinguish, with high probability, between randomly generated tuples of these two types with high probability. Formally:

Definition 2.4. A distinguishing algorithm $\mathcal{D}$ is said to (t, ε)-solve the DDH problem in group $\mathbb{G}$ if $\mathcal{D}$ runs in time at most t and furthermore

$$|\Pr[x, y, z \leftarrow \mathbb{Z}_q\colon \mathcal{D}(g, g^x, g^y, g^z) = 1] - \Pr[x, y \leftarrow \mathbb{Z}_q\colon \mathcal{D}(g, g^x, g^y, g^{xy}) = 1]| \geq \varepsilon.$$

We say that $\mathbb{G}$ is a (t, ε)-DDH group if no algorithm (t, ε)-solves the DDH problem in $\mathbb{G}$.

It is not hard to see that if $\mathbb{G}$ is a (t, ε)-DDH group, then it is a (t', ε')-CDH group for $t' \approx t$ and $\varepsilon' \approx \varepsilon$; that is, hardness of the DDH problem for $\mathbb{G}$ implies hardness of the CDH problem in that group as well. Furthermore, if the CDH problem is hard in $\mathbb{G}$, then the discrete logarithm problem must be hard in $\mathbb{G}$. The converse of these statements is not believed to be true in general. Indeed, there are groups for which the DDH problem is "easy," yet the CDH and discrete logarithm problems in the group are still believed to be hard [25]. On the other hand, for a number of groups of cryptographic interest, "*the best known algorithm for DDH is a full discrete log algorithm*" [5]. These include the commonly used group $\mathbb{G} \subset \mathbb{Z}_p^*$ of order q, where $p = \alpha q + 1$ and p, q are prime with $\gcd(\alpha, q) = 1$. Additionally, Shoup [32] shows that the DDH problem is as hard as the

discrete logarithm problem for *generic* group algorithms (i.e., those that do not use the underlying group structure; see footnote 3). For more details, the reader is referred to [5] and [27].

2.3. *Proving Equality of Discrete Logarithms*

Here, we more carefully consider the soundness property of the protocol given in Section 1.3 for proving equality of discrete logarithms. We also work in a slightly more general setting.

Let $\mathbb{G}$ be a group of prime order q. Slightly generalizing the scenario described earlier, assume values $g, h, y_1 \in \mathbb{G}$ are known to both prover and verifier, and the honest prover knows x such that $g^x = y_1$. (We remark that we do not assume that any of g, h, y_1 are generators of $\mathbb{G}$.) We now let the protocol begin by having the prover send an element y_2 to the verifier. For the honest prover, y_2 will be equal to h^x. The parties then do the following:

1. The prover chooses random $r \in \mathbb{Z}_q$ and sends $A = g^r,\ B = h^r$ to the verifier.
2. The verifier sends to the prover a random $c \in \{0,1\}^k$, where $2^k \leq q$ and c is interpreted as an integer in $\{0, \ldots, q-1\}$.
3. The prover replies with $s = cx + r \bmod q$.
4. The verifier accepts if and only if $A \stackrel{?}{=} g^s y_1^{-c}$ and $B \stackrel{?}{=} h^s y_2^{-c}$.

We stress that the verifier does not check that any of y_2, A, or B are elements of $\mathbb{G}$.

We begin by formally stating the traditional soundness property satisfied by this protocol.

Lemma 1. *Assume $y_2 \in \mathbb{G}$, but there is no x with $g^x = y_1$ and $h^x = y_2$. Then for any A, B sent by a cheating prover, there is at most one value of c for which the verifier will accept.*

Proof. For any c, s, the values $g^s y_1^{-c}$ and $h^s y_2^{-c}$ are elements of $\mathbb{G}$. Hence the verifier cannot possibly accept unless $A, B \in \mathbb{G}$. We assume this from now on.

Say $A, B \in \mathbb{G}$ are such that the prover can send correct responses $s_1, s_2 \in \mathbb{Z}_q$ to two different challenges $c_1, c_2 \in \{0,1\}^k$. Then

$$A = g^{s_1} y_1^{-c_1} = g^{s_2} y_1^{-c_2} \quad \text{and} \quad B = h^{s_1} y_2^{-c_1} = h^{s_2} y_2^{-c_2}.$$

Noting that $c_1 - c_2 \neq 0 \bmod q$, we have

$$g^{(s_1 - s_2)\cdot(c_1 - c_2)^{-1} \bmod q} = y_1 \quad \text{and} \quad h^{(s_1 - s_2)\cdot(c_1 - c_2)^{-1} \bmod q} = y_2,$$

contrary to the assumption of the lemma. □

The above requires that y_2 be an element of $\mathbb{G}$, and typically it is simply assumed that the verifier performs a group membership test to determine whether this is indeed the case. For application to our first signature scheme, however, we want to avoid the computational overhead of performing this test and so we generalize the preceding lemma to the case when $y_2 \notin \mathbb{G}$. In order for operations involving y_2 to be well-defined, we

need to impose some limitations on $\mathbb{G}$; these are necessary for the proof of security of our first signature scheme as well. We remark, though, that the restrictions we impose are fairly mild.

In the remainder of this section, we assume that $\mathbb{G}$ is a subgroup of prime order q of a finite abelian group $\mathbb{H}$ with $|\mathbb{H}| = \alpha \cdot q$ and $q \nmid \alpha$. (For a concrete example, consider the case where $\mathbb{G}$ is the order-q subgroup of $\mathbb{Z}_p^*$ with $p = \alpha q + 1$ and p prime.) By the fundamental theorem for finite abelian groups, this means that $\mathbb{H}$ is isomorphic to $\mathbb{G} \times \mathbb{G}'$ for some abelian group $\mathbb{G}'$ with $|\mathbb{G}'| = \alpha$. For arbitrary $h \in \mathbb{H}$, define the *projection of h onto $\mathbb{G}$* by $\mathsf{proj}_{\mathbb{G}}(h) \stackrel{\text{def}}{=} (h^{\alpha})^{\alpha^{-1} \bmod q}$. It is easy to verify that $\mathsf{proj}_{\mathbb{G}}(h) \in \mathbb{G}$ for any h; $\mathsf{proj}_{\mathbb{G}}(h^a) = \mathsf{proj}_{\mathbb{G}}(h)^a$; and if $h \in \mathbb{G}$ then $\mathsf{proj}_{\mathbb{G}}(h) = h$.

We also modify slightly the protocol described earlier: we now require the verifier to test that $y_2 \in \mathbb{H}$. For our application (and cryptographically appropriate $\mathbb{G}$, $\mathbb{H}$) this will typically be more efficient than testing whether $y_2 \in \mathbb{G}$. The appropriate analogue of the previous lemma follows:

Lemma 2. *Let $g, h, y_1 \in \mathbb{G}$ and $y_2 \in \mathbb{H}$. Assume there is no x with $g^x = y_1$ and $h^x = \mathsf{proj}_{\mathbb{G}}(y_2)$. Then for any A, B sent by a cheating prover, there is at most one value of c for which the verifier will accept.*

Proof. For any c, s, the value $g^s y_1^{-c}$ is an element of $\mathbb{G}$, and $h^s y_2^{-c}$ is an element of $\mathbb{H}$. Hence the verifier will not possibly accept unless $A \in \mathbb{G}$ and $B \in \mathbb{H}$. We assume this from now on.

Say $A \in \mathbb{G}$ and $B \in \mathbb{H}$ are such that the prover can send correct responses s_1, s_2 to two different challenges $c_1, c_2 \in \{0,1\}^k$. Then

$$A = g^{s_1} y_1^{-c_1} = g^{s_2} y_1^{-c_2} \quad \text{and} \quad B = h^{s_1} y_2^{-c_1} = h^{s_2} y_2^{-c_2},$$

and so

$$g^{s_1 - s_2} = y_1^{c_1 - c_2} \quad \text{and} \quad h^{s_1 - s_2} = y_2^{c_1 - c_2}.$$

Since $g, y_1 \in \mathbb{G}$, we see that $g^{(s_1 - s_2) \cdot (c_1 - c_2)^{-1} \bmod q} = y_1$. Taking the projection of both sides of the second equation, and using the fact that $h^{s_1 - s_2} \in \mathbb{G}$, we obtain

$$h^{s_1 - s_2} = \mathsf{proj}_{\mathbb{G}}(h^{s_1 - s_2}) = \mathsf{proj}_{\mathbb{G}}(y_2^{c_1 - c_2}) = \mathsf{proj}_{\mathbb{G}}(y_2)^{c_1 - c_2}.$$

Since $\mathsf{proj}_{\mathbb{G}}(y_2) \in \mathbb{G}$, we conclude that $h^{(s_1 - s_2) \cdot (c_1 - c_2)^{-1} \bmod q} = \mathsf{proj}_{\mathbb{G}}(y_2)$, contrary to the assumption of the lemma. □

2.4. *Random Oracles Mapping to Groups*

Our first scheme requires a random oracle H' mapping its inputs to elements in a group $\mathbb{G}$. However, we would like to assume as a basic primitive only a random oracle H mapping its inputs to bit-strings of some particular length, since standard cryptographic hash functions output bit-strings, not group elements. We discuss a way to construct an H' as desired in general, and then look at two specific examples.

Consider the general case of constructing a random oracle H' mapping to the range $\mathcal{Y}$, using as a building block a random oracle H mapping to the range $\mathcal{X}$. To construct

H' from H, we take a deterministic function $f\colon \mathcal{X} \to \mathcal{Y}$ and define $H'(x) \stackrel{\text{def}}{=} f(H(x))$. In order to prove this construction secure, it suffices to show an efficient simulator Sim that can simulate an adversary's access to H given access to H'; that is, roughly speaking, given a random output $y \in \mathcal{Y}$ from H', the simulator should be able to find a random $x \in \mathcal{X}$ (supposedly output by H) such that $f(x) = y$. Formally, we require the distributions

$$\{y \leftarrow \mathcal{Y};\ x \leftarrow \mathsf{Sim}(y)\colon (x, y)\} \quad \text{and} \quad \{x \leftarrow \mathcal{X}\colon (x, f(x))\}$$

to be statistically indistinguishable. Note in particular that this implies f is invertible (and furthermore it should be possible to choose a random element of $f^{-1}(y)$), and also implies that $f(H(w))$ is close-to-uniformly distributed when $H(w)$ is uniformly distributed.

As an example, consider (as in the previous section) the case where $\mathbb{G}$ is an order-q subgroup (q prime) of an abelian group $\mathbb{H}$ with $|\mathbb{H}| = \alpha q$ and $q \nmid \alpha$. We also assume that it is possible to sample uniformly from elements of $\mathbb{H}$. Given a random oracle H mapping to $\mathbb{H}$, we can construct a random oracle H' mapping to $\mathbb{G}$ by setting $H'(w) = (H(w))^{\alpha}$. It is easy to see that $H'(w)$ is uniformly distributed in $\mathbb{G}$ when $H(w)$ is uniformly distributed in $\mathbb{H}$. Furthermore, we can define a simulator Sim as follows: $\mathsf{Sim}(g)$ (for $g \in \mathbb{G}$) chooses random $h \in \mathbb{H}$ and outputs $\tilde{h} \stackrel{\text{def}}{=} g^{(\alpha^{-1} \bmod q)} \cdot h^q$. Note that $\tilde{h}$ is uniformly distributed among those elements of $\mathbb{H}$ that satisfy $\tilde{h}^{\alpha} = g$.

As a second example, we consider the case of constructing a random oracle H' mapping to $\mathbb{Z}_p^*$ (for p prime) using as a building block a random oracle H mapping to $\{0,1\}^n$. Assuming[8] $n > |p|$, one simple option is to define $H'(w) \stackrel{\text{def}}{=} (H(w) \bmod (p-1)) + 1$. Setting $n = |p| + k$, it is not hard to show a simulator for which the statistical difference between the relevant distributions (as defined above) is $\approx 2^{-k}$. A disadvantage of this approach is that it results in an extra (additive) factor of $\approx q_h \cdot 2^{-k}$ in the security reduction for any scheme based on H', where q_h is the number of hash queries to H'.

A different approach that lends itself to a *perfect* simulation is as follows. (This approach was suggested to us by an anonymous referee.) Say $2^n = \nu \cdot (p-1) + r$ with $0 < r < p-1$, and view H as mapping onto integers in the range $[1,\ \nu \cdot (p-1) + r]$. By reducing modulo $p-1$ and adding 1 as before, we obtain a distribution over $\mathbb{Z}_p^*$ in which elements in the range $S \stackrel{\text{def}}{=} [2, r+1]$ occur slightly more frequently than elements in the range $\mathbb{Z}_p^* \backslash S$. One can correct for this (slight) bias by computing $H'(w)$ as follows:

1. Compute $x = (H(0w) \bmod (p-1)) + 1$. If $x \notin S$, output x and stop. If $x \in S$, output x and stop with probability γ, but with probability $1-\gamma$ continue to the next iteration.
2. Compute $x' = (H(1w) \bmod (p-1)) + 1$ and output x'.

Setting γ appropriately, the output of the above algorithm will be uniformly distributed in $\mathbb{Z}_p^*$. As the focus of our work is on designing signature schemes given a random oracle

[8] It is easy to extend the length of the output of H using standard techniques; for example, to obtain a random oracle $\hat{H}$ with output length $2n$ one can simply define $\hat{H}(w) = H(0w) \parallel H(1w)$, where "$\parallel$" denotes concatenation.

mapping to $\mathbb{G}$ (and not on implementing such random oracles), we do not dwell on this further.

We remark that, for our particular application, a "full-fledged" random oracle mapping onto $\mathbb{G}$ is not needed. In particular, if we are given a random oracle H mapping uniformly onto an efficiently recognizable, dense *subset* S of $\mathbb{G}$, we can simply use H itself in our first scheme. The proof of security can be modified as follows: when simulating the output of the random oracle (see the proof of Theorem 1) the simulator will repeatedly sample random group elements until it finds one that lies in S. This increases the running time of the simulator, but not its success probability.

3. A Signature Scheme Based on the CDH Problem

The scheme we present here was described informally in Section 1.3, and we provide a formal description here. The scheme may be defined over any group $\mathbb{G}$ of prime order q with generator g, though we assume (as discussed in Section 2.3) that $\mathbb{G}$ is an order-q subgroup of an abelian group $\mathbb{H}$ with $|\mathbb{H}| = \alpha \cdot q$ and $q \nmid \alpha$. This assumption can be removed by slightly modifying the scheme; see below.

In the description that follows, we assume for simplicity that $\mathbb{H}, \mathbb{G}, q, \alpha$, and g are publicly known and fixed; alternately, they may be computed during key generation and included in the signer's public key. We let H': $\{0,1\}^* \to \mathbb{G}$ and H: $\{0,1\}^* \to \{0,1\}^k$ be hash functions that will be modeled as random oracles (refer to Section 2.4 for a discussion on constructing H').

Key generation Gen**:** Choose a random $x \leftarrow \mathbb{Z}_q$ and compute $y_1 = g^x$. The public key is y_1 and the secret key is x.

Signature generation $\mathsf{Sign}_{SK}(m)$**:** If m has been signed before, output the previously generated signature (below, we discuss some simple ways to avoid maintaining any state). Otherwise:

1. Choose a random bit b.
2. Compute $h = H'(b, m)$ and $y_2 = h^x$.
3. Generate a non-interactive proof π that (g, h, y_1, y_2) is a Diffie–Hellman tuple. Specifically:
   (a) Choose random $r \leftarrow \mathbb{Z}_q$.
   (b) Compute $A = g^r$, $B = h^r$, and "challenge" $c = H(h, y_2, A, B, m)$.
   (c) Compute $s = cx + r \bmod q$ and set $\pi = (c, s)$.

The signature is (y_2, π, b).

Signature verification $\mathsf{Vrfy}_{PK}(m, \sigma)$**:** Let $PK = y_1$ and parse σ as $(y_2, \pi = (c, s), b)$ where $c \in \{0,1\}^k$, $s \in \mathbb{Z}_q$, $b \in \{0,1\}$, and $y_2 \in \mathbb{H}$. Then:

1. Compute $h = H'(b, m)$.
2. Compute $A = g^s y_1^{-c}$ and $B = h^s y_2^{-c}$.

Output accept if and only if $c \stackrel{?}{=} H(h, y_2, A, B, m)$.

The scheme can be defined over an arbitrary group $\mathbb{G}$ by checking whether $y_2 \in \mathbb{G}$ during signature verification. Depending on the exact group under consideration, this group membership test may impose significant additional cost.

Efficiency improvements. To avoid having the signer maintain a record of all previous message/signature pairs, we can have the signer generate b_m and r as (deterministic) *pseudorandom* functions of the message m; this will result in the same signature being generated each time a particular message is signed. Since we are working in the random oracle model, the simplest implementation of this approach is to set $(b_m, r) = G(SK, m)$ where G is a random oracle independent from H and H'. A modified proof shows that there is essentially no loss in the security reduction by doing this: the only effect on the proof below occurs in case the adversary makes the query $G(SK, \star)$, but we can modify the algorithm below so that it explicitly checks all G-queries of the adversary and thus if the adversary ever makes such a query the algorithm learns x (and can then easily solve the CDH problem).

It is easy to see that the scheme is correct: since $y_1 = g^x$ and $y_2 = h^x$, the verification algorithm computes $A = g^s y_1^{-c} = g^{s-xc} = g^r$, which is the same as the value of A used by the signer (and similarly for B); thus, $H(h, y_2, A, B, m) = c$ and verification outputs accept. We now prove security.

Theorem 1. *Let $\mathbb{G}$ be as above, and assume $\mathbb{G}$ is a (t', ε')-CDH group such that exponentiation in $\mathbb{G}$ takes time t_1 and simulating a query to H' (in the sense described in Section 2.4) takes time t_2. Then the above signature scheme is $(t, q_h, q_s, \varepsilon)$-secure in the sense of unforgeability (in the random oracle model) for*

$$t \approx t' - \mathcal{O}((q_h + q_s) \cdot (t_1 + t_2))$$
$$\varepsilon = 2\varepsilon' + (q_h + 1)/2^k.$$

(*The bound on t is approximate because we do not count operations that are dominated by group exponentiations.*)

Proof. Assume we have an algorithm $\mathcal{F}$ that runs in time at most t, makes at most q_h hash queries (to either H or H') and at most q_s signing queries, and outputs a valid signature on a previously unsigned message with probability at least ε. We use $\mathcal{F}$ to construct an algorithm $\mathcal{A}$ running in time $\approx t'$ that solves the CDH problem with probability at least ε'. The stated result follows immediately since $\mathbb{G}$ is a (t', ε')-CDH group.

Recall we assume that $\mathbb{H}$, $\mathbb{G}$, q, α, and a generator g of $\mathbb{G}$ are fixed and publicly known. Algorithm $\mathcal{A}$ is given as input $(h, y_1) \in \mathbb{G}^2$; setting $x = \log_g y_1$ (which is unknown to $\mathcal{A}$), the goal of $\mathcal{A}$ is to compute h^x. $\mathcal{A}$ sets $PK = y_1$ and runs $\mathcal{F}$ on input PK.

For a message m, we say queries $H'(\star, m)$, $\mathsf{Sign}_{SK}(m)$, and $H(\star, \star, \star, \star, m)$ are *relevant for m*. Any time the first relevant query for some message m is made, $\mathcal{A}$ performs the following steps before answering the query:

1. Choose a random bit b_m and store (b_m, m).
2. Choose random $\gamma_m \in \mathbb{Z}_q$, define $h_m = H'(b_m, m) = g^{\gamma_m}$, and store (b_m, m, γ_m).

3. Compute $y_2 = y_1^{\gamma_m}$ and simulate a non-interactive proof—as discussed in Section 1.3—that $h_m^x = y_2$. (Note that we do not assume $h_m \neq 1$.) Namely, choose random $c \in \{0,1\}^k$ and $s \in \mathbb{Z}_q$ and compute $A = g^s y_1^{-c}$ and $B = h_m^s y_2^{-c}$. Set $\sigma_m = (y_2, c, s, b_m)$, define $H(h_m, y_2, A, B, m) = c$, and store $(\mathsf{sig}, m, \sigma_m)$.

The above steps ensure that $\mathcal{A}$ always has a valid signature for any message m for which a relevant query has been asked.

We now describe how $\mathcal{A}$ simulates the signing and hash oracles for $\mathcal{F}$:

Queries to H'. In response to a query $H'(b, m)$, algorithm $\mathcal{A}$ first checks if the output of H' on this input has been defined previously; note that this is always the case if $b = b_m$ because of the steps performed by $\mathcal{A}$ when the first relevant query for m was made. If so, $\mathcal{A}$ returns the previously assigned value. Otherwise, $b = \bar{b}_m$, and $\mathcal{A}$ chooses random $\beta_m \in \mathbb{Z}_q$, returns $h \cdot g^{\beta_m}$ as the hash output, and stores $(\bar{b}_m, m, \beta_m)$.

Queries to H. In response to a query $H(\star)$, algorithm $\mathcal{A}$ first checks if the output of H on this input has been previously defined. If so, $\mathcal{A}$ returns the previously assigned value. Otherwise, $\mathcal{A}$ responds with a value chosen uniformly at random from $\{0,1\}^k$.

Signing queries. If $\mathcal{F}$ asks for a signature on a message m, algorithm $\mathcal{A}$ finds the stored tuple of the form $(\mathsf{sig}, m, \star)$; a unique tuple of this form exists (by construction of $\mathcal{A}$) and we let σ_m denote the value of the final element in this tuple. $\mathcal{A}$ returns σ_m.

At some point, $\mathcal{F}$ outputs its supposed forgery $(\hat{m}, \hat{\sigma} = (\hat{y}_2, \hat{c}, \hat{s}, \hat{b}))$, where $\mathcal{F}$ did not previously request a signature on $\hat{m}$. Algorithm $\mathcal{A}$ checks whether: (1) $\mathsf{Vrfy}_{PK}(\hat{m}, \hat{\sigma}) = \mathsf{accept}$, and (2) $\hat{b} = \bar{b}_{\hat{m}}$. (Note that $\mathcal{A}$ may be required to simulate additional queries to H, H' in case $\mathcal{F}$ did not make the relevant queries itself.) If either of these do not hold, then $\mathcal{A}$ simply aborts. Otherwise, $\mathcal{A}$ finds the stored tuple of the form $(\bar{b}_{\hat{m}}, \hat{m}, \star)$; a unique such tuple exists (by construction of $\mathcal{A}$), and we let $\beta_{\hat{m}}$ denote the value of the final element in this tuple. $\mathcal{A}$ outputs $\mathsf{proj}_{\mathbb{G}}(\hat{y}_2)/y_1^{\beta_{\hat{m}}}$ (see Section 2.3 for a definition of $\mathsf{proj}_{\mathbb{G}}(\cdot)$, and note that $\mathcal{A}$ can compute this since α is known). This completes the description of $\mathcal{A}$.

We first claim that $\mathcal{A}$ provides a simulation for $\mathcal{F}$ whose distribution is identical to the distribution on the view of $\mathcal{F}$ in a real interaction with a signer. To see this, note that:

1. Since g is a generator of $\mathbb{G}$, the output of any query $H'(b, m)$—regardless of whether $b = b_m$ or not—is uniformly distributed in $\mathbb{G}$ as required.
2. The simulation of the H oracle is obviously perfect.
3. Consider the signature $\sigma = (y_2, c, s, b_m)$ returned by $\mathcal{A}$ in response to a signing query $\mathsf{Sign}_{SK}(m)$. Note that this signature is constructed by $\mathcal{A}$ at the time the first relevant query for m was made. Clearly, b_m is uniformly distributed. Letting $h_m = H'(b_m, m) = g^{\gamma_m}$, we see that $y_2 = y_1^{\gamma_m} = (g^x)^{\gamma_m} = h_m^x$, just as in the real experiment. Finally, (c, s) is distributed as in the real experiment by the honest-verifier zero-knowledge property of the proof system.

Thus, the probability that $\mathcal{F}$ outputs a valid forgery in the simulated experiment is exactly ε.

Assume $\mathcal{F}$ outputs a valid forgery $(\hat{m}, (\hat{y}_2, \hat{c}, \hat{s}, \hat{b}))$, and let $\hat{h} = H'(\hat{b}, \hat{m})$. We argue that, with all but negligible probability, $\mathsf{proj}_{\mathbb{G}}(\hat{y}_2) = \hat{h}^x$; if so, say $\hat{y}_2$ is *good*. Indeed, if $\hat{y}_2$ is not good then (using Lemma 2) for any A, B there is at most one possible value of c for which there exists an s satisfying $A = g^s y_1^{-c}$ and $B = \hat{h}^s \hat{y}_2^{-c}$. If $\hat{y}_2$ is not good, then, for any hash query $H(\hat{h}, \hat{y}_2, A, B, \hat{m})$ made by $\mathcal{F}$ the probability that the query returns a c for which there exists an s as above is at most $1/2^k$. It follows that the probability that $\mathcal{F}$ outputs a valid forgery where $\hat{y}_2$ is not good is at most $(q_h + 1)/2^k$. (The additive factor of 1 occurs in case $\mathcal{F}$ did not query $H(\hat{h},\ \hat{y}_2,\ g^{\hat{s}} y_1^{-\hat{c}},\ \hat{h}^{\hat{s}} \hat{y}_2^{-\hat{c}},\ \hat{m})$ itself.) We conclude that in the simulated experiment described above, $\mathcal{F}$ outputs a valid forgery where $\hat{y}_2$ is good with probability at least $\varepsilon - (q_h + 1)/2^k$.

Now, since $\mathcal{F}$ did not previously request a signature on $\hat{m}$, the value of $b_{\hat{m}}$ is independent of the view of $\mathcal{F}$. So, the probability that $\mathcal{F}$ outputs a valid forgery such that $\hat{y}_2$ is good and also $\hat{b} = \bar{b}_{\hat{m}}$ is at least $1/2 \cdot (\varepsilon - (q_h + 1)/2^k)$. To finish the proof, we claim that whenever this event occurs $\mathcal{A}$ outputs the correct solution to its given instance of the CDH problem. To see this, note that when $\hat{y}_2$ is good the output of $\mathcal{A}$ satisfies

$$\begin{aligned} \mathsf{proj}_{\mathbb{G}}\left(\hat{y}_2\right) / y_1^{\beta_{\hat{m}}} &= \hat{h}^x / y_1^{\beta_{\hat{m}}} \\ &= (h g^{\beta_{\hat{m}}})^x / y_1^{\beta_{\hat{m}}} = h^x, \end{aligned}$$

as desired.

We conclude that $\mathcal{A}$ outputs the correct solution with probability at least $1/2 \cdot (\varepsilon - (q_h + 1)/2^k) \geq \varepsilon'$. Examining the running time of $\mathcal{A}$ gives the result of the theorem. □

We remark that the constant term in the expression for t in the theorem can be improved by modifying the proof so that a signature on a message m is not computed by $\mathcal{A}$ at the time of the first relevant query for m, but is instead computed by $\mathcal{A}$ only when $\mathcal{F}$ actually requests a signature on m. This results in a slightly worse bound on ε (though the difference is unimportant). We have decided not to present the proof in this way because we believe the current proof is conceptually simpler, and anyway the bound on t is only approximate since we do not take into account the complexity of all operations of $\mathcal{A}$.

4. A Signature Scheme Based on the DDH Problem

In the previous scheme, a message m is signed by mapping m to some group element h, computing $y_2 = h^x$, and then proving that (g, h, y_1, y_2) is a Diffie–Hellman tuple. Distilling out the intuition behind the proof of Theorem 1, we see that previous scheme is secure under the CDH assumption since—given some h associated with a previously unsigned message m—an adversary "must" produce $y_2 = h^x$ in order to generate a convincing proof that (g, h, y_1, y_2) is a Diffie–Hellman tuple (we ignore here the possibility that $y_2 \notin \mathbb{G}$, which can be taken into account as described in the proof of Theorem 1).

We notice here that if one is willing to base security of the scheme on the (stronger) DDH assumption, it is unnecessary to generate a new h for each message; instead, the public key can simply contain the fixed tuple (g, h, y_1, y_2) and a signature will consist

of a proof that this is a Diffie–Hellman tuple. The resulting scheme is more efficient than the previous scheme.

As before, we assume that $\mathbb{G}$ is a cyclic group of prime order q with generator g; in contrast to the previous section, $\mathbb{G}$ is otherwise completely arbitrary. Let $H\colon \{0,1\}^* \to \{0,1\}^k$ be a hash function that will be modeled as a random oracle. Our second scheme is defined as follows:

Key generation Gen**:** Choose a random $h \in \mathbb{G}$ and a random value $x \leftarrow \mathbb{Z}_q$. Compute $y_1 = g^x$ and $y_2 = h^x$. The public key is the Diffie–Hellman tuple $PK = (h, y_1, y_2)$ and the secret key is x.

Signature generation $\mathsf{Sign}_{SK}(m)$**:** If m has been signed before, output the previously generated signature (we can avoid maintaining state exactly as discussed in the previous section). Otherwise, generate a non-interactive proof—depending on m—that the public key is a Diffie–Hellman tuple. Specifically:

1. Choose random $r \leftarrow \mathbb{Z}_q$.
2. Compute $A = g^r$, $B = h^r$, and "challenge" $c = H(A, B, m)$.
3. Compute $s = cx + r \bmod q$ and set $\pi = (c, s)$.

The signature is π.

Signature verification $\mathsf{Vrfy}_{PK}(m, \sigma)$**:** Parse PK as (h, y_1, y_2) and σ as $\pi = (c, s)$ where $c \in \{0,1\}^k$ and $s \in \mathbb{Z}_q$. Compute $A = g^s y_1^{-c}$ and $B = h^s y_2^{-c}$; output accept if and only if $c \stackrel{?}{=} H(A, B, m)$.

We remark that, in contrast to the previous scheme, the present scheme can be proven secure even if the same message is signed multiple times using independent randomness (i.e., it is not strictly necessary for the signing algorithm to check whether the given message was signed previously). We have chosen to present the scheme as we did because the security reduction is slightly[9] tighter; the proof is a bit simpler; and the overhead of making signature generation deterministic is not (in general) significant.

It is not hard to see that the scheme is correct. We now prove security.

Theorem 2. *Let $\mathbb{G}$ be as above, and assume $\mathbb{G}$ is a (t', ε')-DDH group such that exponentiation in $\mathbb{G}$ takes time t_1. Then the above signature scheme is $(t, q_h, q_s, \varepsilon)$-secure in the sense of strong unforgeability (in the random oracle model) for*

$$\begin{aligned} t &\approx t' - \mathcal{O}(q_s \cdot t_1), \\ \varepsilon &= \varepsilon' + (q_h + 1)/q + (q_h + 1)/2^k. \end{aligned}$$

(*The bound on t is approximate because we do not count operations that are dominated by group exponentiations.*)

[9] If messages are signed multiple times, the same reduction as in the proof of the following theorem gives $\varepsilon = \varepsilon' + (q_s q_h + 1)/q + (q_h + 1)/2^k$.

Proof. Assume we have an algorithm $\mathcal{F}$ that runs in time at most t, makes at most q_h hash queries and at most q_s signing queries, and outputs a new, valid message/signature pair with probability at least ε. We use $\mathcal{F}$ to construct an algorithm $\mathcal{D}$ running in time $\approx t'$ which solves the DDH problem with probability ε'. The stated result follows immediately since $\mathbb{G}$ is a (t', ε')-DDH group.

Algorithm $\mathcal{D}$ is given as input a tuple (g, h, y_1, y_2); its goal, informally, is to determine whether this is a random tuple or a Diffie–Hellman tuple (see Section 2.2). To this end, it sets $PK = (h, y_1, y_2)$ and runs $\mathcal{F}$ on input PK. Algorithm $\mathcal{D}$ simulates the signing and hash oracle for $\mathcal{F}$ as follows:

Hash queries. In response to a query $H(A, B, m)$, algorithm $\mathcal{D}$ first checks if the output of H on this input has been previously defined (either directly by a previous hash query or as part of a signature query). If so, $\mathcal{D}$ returns the previously assigned value. Otherwise, $\mathcal{D}$ responds with a value chosen uniformly at random from $\{0, 1\}^k$.

Signing queries. When $\mathcal{F}$ asks for a signature on message m, algorithm $\mathcal{D}$ first checks if this message was signed before; if so, $\mathcal{D}$ outputs the previously generated signature. Otherwise, $\mathcal{D}$ attempts to simulate a proof that (g, h, y_1, y_2) is a DDH tuple as follows: $\mathcal{D}$ chooses random $c \in \{0, 1\}^k$ and $s \in \mathbb{Z}_q$, and computes $A = g^s y_1^{-c}$ and $B = h^s y_2^{-c}$. If H had previously been queried on input (A, B, m) and $H(A, B, m) \neq c$, then $\mathcal{D}$ aborts (and outputs 0); otherwise, $\mathcal{D}$ sets $H(A, B, m) = c$ (if it was not set this way already) and outputs the signature (c, s)

At some point, $\mathcal{F}$ outputs its forgery $(\tilde{m}, \tilde{\sigma} = (\tilde{c}, \tilde{s}))$ where $\tilde{\sigma}$ was not previously the response to a query $\mathsf{Sign}_{SK}(\tilde{m})$. If $\mathsf{Vrfy}_{PK}(\tilde{m}, \tilde{\sigma}) = 1$, then $\mathcal{D}$ outputs 1; otherwise, $\mathcal{D}$ outputs 0. (Note that verifying the signature may require $\mathcal{D}$ to simulate an additional query to H.)

We first analyze the probability that $\mathcal{D}$ outputs 1 when (g, h, y_1, y_2) is a Diffie–Hellman tuple. In this case, $\mathcal{D}$ provides a simulation for $\mathcal{F}$ whose distribution is statistically close to the distribution from the view of $\mathcal{F}$ during a real interaction with a signer, with the only difference arising in case $\mathcal{A}$ aborts when answering a signing query. When answering any particular query $\mathsf{Sign}_{SK}(m)$, the probability that $\mathcal{D}$ aborts is at most $q_h(m)/q$, where $q_h(m)$ is the number of H-queries made by $\mathcal{F}$ of the form $H(\star, \star, m)$. Furthermore, this only applies the *first* time this signature query is made, as $\mathcal{D}$ simply outputs the previous signature if another signature is requested on the same message m. As in the proof of Theorem 1, we can upper-bound the probability that $\mathcal{D}$ aborts by q_h/q. It follows that $\mathcal{F}$ outputs a valid forgery (and hence $\mathcal{D}$ outputs 1) with probability at least $\varepsilon - q_h/q$.

On the other hand, if (g, h, y_1, y_2) is a random tuple, then it is *not* a Diffie–Hellman tuple with probability $1 - 1/q$. In this case, for any A, B and any query $H(A, B, m)$ made by $\mathcal{F}$ it follows from Lemma 1 that there is at most one possible value of c for which there exists an s satisfying $A = g^s y_1^{-c}$ and $B = h^s y_2^{-c}$. Thus, $\mathcal{F}$ outputs a forgery (and hence $\mathcal{D}$ outputs 1) with probability at most $1/q + (q_h + 1)/2^k$. (As in the previous proof, the additive factor of 1 occurs in case $\mathcal{F}$ did not make the relevant H-query for its forgery.)

Putting everything together, we see that

$$\begin{aligned}|\Pr[x, y \leftarrow \mathbb{Z}_q\colon \mathcal{D}(g, g^x, g^y, g^{xy}) = 1] - \Pr[x, y, z \leftarrow \mathbb{Z}_q\colon \mathcal{D}(g, g^x, g^y, g^z) = 1]| \\ \geq \varepsilon - (q_h + 1)/q - (q_h + 1)/2^k \\ \geq \varepsilon'.\end{aligned}$$

Examining the running time of $\mathcal{D}$ gives the result of the theorem. □

Acknowledgments

We thank Benoît Chevallier-Mames and the anonymous referees for their helpful comments on earlier drafts of this paper.

References

[1] *Advances in Cryptology—Crypto* '89. Volume 435 of Lecture Notes in Computer Science. Springer, Berlin, 1990.

[2] *Advances in Cryptology—Crypto* 2005. Volume 3621 of Lecture Notes in Computer Science. Springer, Berlin, 2005.

[3] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, New York, 1993.

[4] M. Bellare and P. Rogaway. The exact security of digital signatures—how to sign with RSA and Rabin. In *Advances in Cryptology—Eurocrypt* '96, pages 399–416. Volume 1070 of Lecture Notes in Computer Science. Springer, Berlin, 1996.

[5] D. Boneh. The decision Diffie-Hellman problem. In *Algorithmic Number Theory*, 3*rd International Symposium*, pages 48–63. Volume 1423 of Lecture Notes in Computer Science. Springer, Berlin, 1998.

[6] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[7] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *Advances in Cryptology—Asiacrypt* 2001, pages 514–532. Volume 2248 of Lecture Notes in Computer Science. Springer, Berlin, 2001.

[8] E. Brickell, D. Pointcheval, S. Vaudenay, and M. Yung. Design validations for discrete logarithm based signature schemes. In *Public-Key Cryptography*, pages 276–292. Volume 1751 of Lecture Notes in Computer Science. Springer, Berlin, 2000.

[9] J. Camenisch and M. Stadler. Proof systems for general statements about discrete logarithms. Technical Report 260, Institute for Theoretical Computer Science, ETH Zurich, 1997.

[10] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[11] D. Chaum and H. Van Antwerpen. Undeniable signatures. In *Advances in Cryptology—Crypto* '89 [1], pages 212–216.

[12] D. Chaum, J.-H. Evertse, and J. van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In *Advances in Cryptology—Eurocrypt* '87, pages 127–142. Volume 304 of Lecture Notes in Computer Science. Springer, Berlin, 1988.

[13] D. Chaum and T. Pedersen. Wallet databases with observers. In *Advances in Cryptology—Crypto* '92, pages 89–105. Volume 740 of Lecture Notes in Computer Science. Springer, Berlin, 1993.

[14] B. Chevallier-Mames. An efficient CDH-based signature scheme with a tight security reduction. In *Advances in Cryptology—Crypto* 2005 [2], pages 511–526.

[15] J.-S. Coron. On the exact security of full-domain hash. In *Advances in Cryptology—Crypto* 2000, pages 229–235. Volume 1880 of Lecture Notes in Computer Science. Springer, Berlin, 2000.

[16] J.-S. Coron. Optimal security proofs for PSS and other signature schemes. In *Advances in Cryptology—Eurocrypt* 2002, pages 272–287. Volume 2332 of Lecture Notes in Computer Science. Springer, Berlin, 2002.
[17] R. Cramer and I. Damgrd. Secure signature schemes based on interactive protocols. In *Advances in Cryptology—Crypto* '95, pages 297–310. Volume 963 of Lecture Notes in Computer Science. Springer, Berlin, 1995.
[18] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, 22(6):644–654, 1976.
[19] Y. Dodis and L. Reyzin. On the power of claw-free permutations. In *Security in Communication Networks* (*SCN* 2002), pages 55–73. Volume 2576 of Lecture Notes in Computer Science. Springer, Berlin, 2003.
[20] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology—Crypto* '86, pages 186–194. Volume 263 of Lecture Notes in Computer Science. Springer, Berlin, 1987.
[21] M. Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *Advances in Cryptology—Crypto* 2005 [2], pages 152–168.
[22] T. El Gamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, 31(4):469–472, 1985.
[23] E.-J. Goh and S. Jarecki. A signature scheme as secure as the Diffie–Hellman problem. In *Advances in Cryptology—Eurocrypt* 2003, volume 2656 of Lecture Notes in Computer Science, pages 401–415. Springer, 2003.
[24] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.
[25] A. Joux and K. Nguyen. Separating decision Diffie–Hellman from Diffie–Hellman in cryptographic groups. Available at http://eprint.iacr.org/2001/003.
[26] J. Katz and N. Wang. Efficiency improvements for signature schemes with tight security reductions. In *Proceedings of the* 10*th ACM Conference on Computer and Communications Security*, pages 155–164, 2003.
[27] U. Maurer and S. Wolf. The Diffie-Hellman protocol. *Des. Codes, Cryptogr.*, 19(2/3):147–171, 2000.
[28] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, 1997.
[29] S. Micali and L. Reyzin. Improving the exact security of digital signature schemes. *J. Cryptology*, 15(1):1–18, 2002.
[30] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–396, 2000.
[31] C.-P. Schnorr. Efficient identification and signatures for smart cards. In *Advances in Cryptology—Crypto* '89 [1], pages 239–252.
[32] V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology—Eurocrypt* '97, pages 256–266. Volume 1233 of Lecture Notes in Computer Science. Springer, Berlin, 1997.
[33] U.S. Department of Commerce/National Institute of Standards and Technology. *Digital Signature Standard*, 2000. Federal Information Processing Standards, publication #186-2.