Signing with Partially Adversarial Hashing

Silvio Micali Leonid Reyzin *

MIT Laboratory for Computer Science
Cambridge MA 02139

MIT-LCS-TM-575

February 27, 1998

Abstract. Digital signatures usually utilize one-way hash functions designed by other parties. It is thus possible that such hash functions are adverserially designed so as to enable forging signatures in otherwise secure schemes.

We initiate the study of signing with adversarial hashing by considering hash functions for which an adversary can fix arbitrarily the input-output values at polynomially many inputs.

On the negative side, we show that many known signature schemes are vulnerable against such partially adversarial hashing attacks. On the positive side, we show that a surprisingly *simple* modification makes any scheme invulnerable against such attacks. The computational requirements of our modification are so mild that it could be easily integrated into any signature scheme.

Keywors: cryptanalysis, cryptography, digital signature, hash function, random oracle

1 Introduction

Typically, a digital signature scheme can directly sign k-bit messages, where the quantity k depends on the security parameter (e.g., the length of the public key). Longer messages are first hashed prior to being signed. That is, if a message M is more than k-bit long, its signature consists of

$$SIG(PK, SK, H(M)),$$

where SIG represents the (possibly probabilistic) signing algorithm, PK and SK are its public and secret keys, and H is an easy to compute hash function mapping the set of all binary strings into k-bit (or shorter) strings. Popular hash function used in conjunction with digital signature schemes include SHA-1 [NIS95], MD5 [Riv92], or RIPEMD [DBP96].

For this hash-and-sign strategy to work, it is necessary that H be a *one-way hash function*, that is, it must be computationally hard to invert H or to find any two strings x and y such that $H(x) = H(y)$, because a valid signature of x also is a valid signature of y. Typically, however, H should satisfy additional requirements, such as "breaking" the algebraic structure of the underlying signature scheme. Ideally, H should be a random oracle mapping strings of arbitrary length to strings of length k.

*Web: `http://theory.lcs.mit.edu/~reyzin`. E-mail: `reyzin@mit.edu`.

This material is based upon work supported in part under a National Science Foundation Graduate Fellowship.

Some digital signature schemes, most notably the Goldwasser, Micali and Rivest scheme [GMR88], implicitly design their own hash functions, and are provably secure without the use of the random oracle assumption. Such schemes, however, are rare and not very efficient. Most of the time digital signature schemes use independently designed hash functions and their provable security depends on the hash functions being random oracles.

1.1 Pros and Cons of Separating Hash-Function and Digital-Signature Design

Pros. There are valid reasons for having digital signature schemes use independently designed hash functions.

First, the technical skills required for designing a good hash function are different from those required for the design of a good signature scheme.

Second, to reap the fruits of standardization, it is preferable to deal with one or a few hash functions and possibly many digital signature schemes, rather than with many signature schemes and as many hash functions.

Third, a hash function usually needs to be trusted by more than one party with conflicting interests. For example, the signer needs to trust that the hash function makes it hard for someone else to forge signatures, and verifiers need to trust that the signer will not be able to later repudiate signatures by claiming that the hash function was weak. Thus, it may not be advisable (let alone practical) that each user design its own hash function.

Cons. Separating the design of hash functions from the design of signature schemes, however, has a main disadvantage: namely, it opens up the possibility that a hash function is adverserialy chosen, so as to enable one to forge signatures in the signature schemes that use it.

Of course, in order to use a hash function H with a given signature scheme, we need some convincing arguments pointing to H's security. Publicly known hash functions tend to receive extensive cryptanalysis which is often able to convince us that they are hard to invert and that it is hard to find collisions or any other interesting relations among sets of inputs and outputs. However,

cryptanalysis may not reveal that the designer built some other secret weakness into the hash function.

For example, to obtain a hash function mapping $2k$-bit strings into k-bit strings, an adversary may choose at random two $2k$-bit strings x and y and a random k-bit string z, and then design H so that $H(x) = H(y) = z$. Therefore, such an adversary knows a collision for H. This, however, does not make finding H-collisions easy for anybody else. In fact, assuming that H is otherwise perfectly designed, no cryptanalysis may reveal that H "behaves strangely" in any way. Formally speaking, this is so because, modeling H as a random oracle, forcing a collision as above does not alter H's probability distribution at all.

Some users of SHA-1 may not worry about any such adversarially-introduced defects, becase SHA-1 has been designed by the U.S. National Security Agency. Other users may instead worry a lot, for the very the same reason. Thus, the question is,

can we safely separate the design of hash functions and signature schemes?

1.2 Contributions of This Paper.

In this paper we initiate the study of signing with adversarial hashing. We do so by considering a simple but realistic model: an adversary that designs a hash function so as to force its input-output relation at a few inputs of its choice without being detected. We call this model *partially adversarial hashing*. In it the adversary is actually allowed to fix H's value at polynomially many (in k) points. These points are more than just "few," which of course make our model stronger. The model makes use of random oracles to ensure that there are no "weaknesses" other than those introduced by the adversarial design. That is, the model guarantees that the so designed hash functions are not "junk" but perform excedingly well so as to withstand any amount of cryptanalysis.

We show that most schemes in the literature are insecure if implemented with partially-adversarial hash functions. This is to be expected, of course, for deterministic hash-and-sign schemes. However, we show that this insecurity extends to *probabilistic* schemes as well, such as the Bellare-Rogaway [BR96] scheme. Because this scheme is otherwise "perfect" (that is, provably secure if one hashes by means of a random oracle), this proves the point that adversarial hashing (even in our simple form) is a serious concern.

More importantly, and more optimistically, we show that all signature schemes secure in the random oracle model can be made secure in partially adversarial hashing model by means of a

conceptually and algorithmically simple fix.

The fix entails the additional hashing of few hundred bits. Such an operation can clearly be incorporated in any signature scheme without much cost, while buying a reasonable amount of extra security, whenever using someone else's hash function.

2 The Partially-Adversarial Hashing Model

Given a signature scheme (K, S, V), where K, S and V are, respectively, the key-generation, signature and verification algorithms, our model consists of a 3-move game between two polynomially-bounded players: a *hash designer* and a *signer*. Because of its adversarial nature, we refer to the hash designer as an adversary (and denote it by A).

1. First, on input (K, S, V) and security parameter k, the adversary A chooses a set of size l of pairs

   $$\{(x_i, y_i) | x_i \in \{0,1\}^*, y_i \in \{0,1\}^k, 1 \leq i \leq l\},$$

   where $l = L(k)$ for some fixed polynomial L.

   Then, a random oracle H is so constructed:

   For x_i $(1 \leq i \leq l)$, $H(x_i) = y_i$; for all other x, $H(x)$ is a random $k-$bit string.

2. The signer runs the algorithm K to choose a random public/secret key pair (PK, SK), announces PK as its public key, (K, S, V) as its signature scheme, and H as the hash function.

3. The adversary, given the pairs (x_i, y_i) as a private input, mounts an adaptive chosen-message attack on the signature scheme (K, S, V) with the public key PK.

A digital signature scheme (K, S, V) is said to be *insecure* against partially adversarial hash functions if the adversary succeeds in existentially forging a signature in step 3 with a polynomially high probability.

Notice that if the pairs (x_i, y_i) are properly chosen, then the distribution of H is exactly that of a random oracle (which is not the case if, for instance, A chooses H so that $H(0) = H(1)$). We do not restrict, however, the adversary to choose its pairs in such a manner, which makes its attack (and thus the positive result of Section 4) stronger.[1]

[1] Of course, having no deviation from the distribution of a random oracle guarantees that all cryptanalysis will fail in revealing the presence of an adversarial design. At the same time, the converse is not necessarily true, because not all such deviations could be easily detected by cryptanalysis. For instance, for $i = 1, \ldots, 100$ and a random string R, consider the strings $x_i = R||i$, that is, R concatenated with the binary representation of i. Then, if for all 100 i, $H(x_i) = i$, this is a strong indication that H has been "fixed," though such an indication may not be easy to find if R is long enough.

It should also be appreciated that our model is quite tangential to the problem of whether random oracles are actually implementable by means of easy to compute functions with public code. In fact, we use random oracles only to model the notion that the adversary could be "caught" because of defects of the hash function other than the forced input-output relations.

Finally, it should also be noted that having the adversary choose H with knowledge of the signature scheme used by the signer is quite realistic. In fact, if some signature scheme attracting much attention arises, the adversary may bet that it will be sooner or later be used. At the same time, it is quite realistic that the adversary does not know the public key of the signature scheme. Indeed, when choosing its public key, a signer knows already which hash function will be used in verifying and producing its signatures. Such function is in fact typically specified when publishing the chosen public key and signature algorithm.

Let us now analyze the security of a few known signature schemes with respect to partially adversarial hashing.

3 Partially Adversarial Hashing Attacks on Known Signature Schemes

3.1 Attacks on deterministic hash-and-sign schemes

THE SIGNATURE SCHEMES. In a typical hash-and-sign scheme, a signature s of a message M is computed as follows: $s = SIG(PK, SK, H(M))$. Examples of such schemes include many variants of RSA, such as [RSA93, IEE97], Rabin and Williams [Rab79, Wil80, IEE97], DSA [NIS94] and Elliptic Curve DSA [IEE97], Schnorr [Sch91], Nyberg-Rueppel with a hash function [NR96, IEE97] and many others. Some such schemes are proven secure (equivalent to problems deemed hard, such as factoring) assuming H is a random oracle (see, for example, [BR93]).

THE ATTACK. To attack a deterministic hash-and-sign scheme, the adversary A designs the hash funciton as follows. A picks a message M_1 whose signature it wants to be able to forge (for example, $M_1 =$"I owe A $1,000,000") and another message M_2 that a signer is more likely to sign (for example, $M_2 =$"I owe A $1"). A then fixes $H(M_1) = H(M_2)$. Assume now that a signer S uses H as its hash function in a deterministic hash-and-sign scheme. Then even though S selects its own PK and SK after H is chosen, by getting S to sign M_2, A also obtains a signature on M_1.

DISCUSSION. While A has to mount a chosen-message attack in order to succeed, this chosen-message attack is fairly easy to mount: A needs to get only one chosen message signed, and the message does not depend on the public key (i.e., A need not be adaptive). Such an attack is feasible in many settings.

Notice that this attack extends to other schemes as well. For instance, if the messages are padded with the current time prior to signing ("time-stamped"), when designing the hash function, A can choose times $s_1, s_2, \ldots, s_i$ and $t_1, t_2, \ldots, t_i$ and fix $H(M_1||s_j) = H(M_2||t_j)$ for $1 \leq j \leq i$ (here "$||$" denotes concatenation). All it needs then is to obtain the signature on M_2 at time t_j for some j.

Neither the attack, nor the messages M_1 and M_2, depend on the signature algorithm used, so a single adversarial hash function can be used to attack all such signature schemes.

3.2 Attacks on probabilistic hash-and-sign schemes

From the previous section, it may seem that only signature schemes that use hash functions deterministically are vulnerable in this model. This is not the case. Consider the following scheme, known as PSS [BR96]. Below we present PSS with exponent-3 RSA (RSA with verification exponent 3 is often used in practice to expedite verification).

THE BELLARE-ROGAWAY PROBABILISTIC SIGNATURE SCHEME (PSS). Since PSS is a complicated scheme, it may be helpful to refer to its description given in [BR96].

- **Setup:** Before executing the scheme, one needs to pick a hash function $H : \{0,1\}^* \rightarrow \{0,1\}^k$, a key length $b > k$, and a generator function $G : \{0,1\}^k \rightarrow \{0,1\}^{b-k-1}$ (in [BR96], both H and G are assumed to be random oracles). One also needs to pick a seed length k_0 (k_0 can be equal to k.) Let $G_0(x)$ be the function which gives the first k_0 bits of $G(x)$, and $G_1(x)$ be the function which gives the remaining $b - k - k_0 - 1$ bits of $G(x)$.

- **Key Generation:** Generate two random large primes p and q such that $p \equiv q \equiv 2 \pmod 3$ and the length in bits of $n = pq$ is b. Compute d such that $3d \equiv 1 \pmod{LCM(p-1, q-1)}$ (here LCM denotes the least common multiple). Output (n, d) as the private key and $(n, 3)$ as the public key.

- **Signing:** To sign a message M, generate a random bit string r of length k_0, compute $w = H(M||r)$, $r^* = G_1(w) \oplus r$ and $u = 0||w||r^*||G_2(w)$ (here "$||$" denotes concatenation of strings and "$\oplus$" denotes bit-wise exclusive-or). Compute $s = u^3 \mod n$ and output s as the signature.

- **Verifying:** To verify a signature s on a message M, compute $u' = s^e \mod n$. Make u' into a bit string of length b (that is, pad it with zeroes on the left if necessary). Let f be the first bit of u', w' be the next k bits, r'^* be the next k_0 bits and y be the remaining bits. Let $r' = r'^* \oplus G_1(w')$. Verify that $f = 0$, $w' = H(M||r')$ and that $y = G_2(w')$. If all of these are true, accept the signature; reject otherwise.

In [BR96], Bellare and Rogaway prove that existentially forging PSS signatures is equivalent to inverting RSA, assuming H and G are random oracles.

THE ATTACK. To attack PSS, the adversary A can design H and G as follows. A picks a message M whose signature it wants to be able to forge (for example, $M =$"I owe A \$1,000,000"), a random positive integer $i < 2^{(b-1)/3}$ and a random bit string r of length k_0. Let $u = i^3$. Make u into a bit string of length $b-1$ (that is, pad it with zeroes on the left if necessary). Let w be the first k bits of u, r^* be the next k_0 bits, and g_2 be the remaining bits. A sets $H(M||r) = w$, $g_1 = r \oplus r^*$ and $G(w) = g_1||g_2$. Then $s = i$ is a valid signature on M. So, A knows a signature on M that is valid regardless of the public key.

DISCUSSION. Note that A does not need to mount a chosen-message attack, but can forge signatures directly. Also note that A does not fix $H(M)$, but rather $H(M||r)$, which makes it harder to detect that H is adversarial.

The same attack can be mounted on PSS with a low exponent other than 3 and on PSS with Rabin instead of RSA (called PRab in [BR96]). (With PRab, where exponent 2 is used instead of 3, there are some techincal differences due to the fact that not every number in ${Z_n}^*$ has a square root, but they do not prevent A from mounting this attack.)

In general, consider a randomized hash-and-sign scheme where to sign a message M, one computes, independent of the keys, some value $u = U^{H_1, H_2, \ldots, H_i}(M, r)$ (here $H_1, H_2, \ldots, H_i$ are hash functions and r is a random string) and then signs u using some signature algorithm SIG and the keys PK and SK: $s = SIG(PK, SK, u)$. As lons as there are values of s and u such that $s = SIG(PK, SK, u)$ independently of the PK and SK, this attack can be mounted.

4 Achieving Security Against Partially Adverserial Hahsing

In this section, we propose a simple and effective change that turns any hash-and-sign signature scheme secure when implemented with a random oracle into a scheme secure when implemented with a partially-adversarial hash function.

THE CHANGE. Any time a hash function H needs to be evaluated on an input x, evaluate $H(PK||x)$ instead.

Notice that the change works independently of the signature scheme, does not change the length of the signatures, and has minimal impact on signature computation.

THE CHANGE WORKS. The reason why the change works is easy to explain informally. Since the underlying signature scheme was proven to be secure assuming that no one could "fix" the values of the random-oracle hash function, the idea is to avoid the inputs to the hash function that were fixed by the adversary. A natural way to do this is to randomize the input to the hash function: as long as there is enough randomness in the input, the values that the adversary fixed will be avoided with very high probability.

The random pad r in PSS (Section 3.2) almost accomplishes that goal. As long as r is long enough, a PSS *signer* will be very unlikely to use either H or G on an input that was fixed by the adversary. Therefore, the adversary cannot mount a chosen-message attack on PSS similar to the one in Section 3.1. However, a PSS *verifier* does not randomize its inputs to H or G—they are determined by the signature received. This makes it possible for the adversary to mount the direct attack presented in Section 3.2.

What we need is some randomness *shared* among the signer and the verifiers that is inaccessible to the adversary when he designs the hash function. Since an adversarial hash function is designed before a signer chooses its public key, the natural choice for this randomness is padding with this very public key.

Such padding avoids with high probability any input at which the adversary fixes H's output. The intuition beyond this is simple: if the public key takes a given value with probablity $p > poly(k)$, then the signature scheme can be easily broken with probability p^2. Indeed, with probability p a signer will choose one such value for its public key, and with the same probability p a forger running the key generator algorithm will output the same public key together with a valid secret key. Whether or not the secret key computed by the signer is the same as that computed by the forger is irrelevant: the forger will be able to sign messages at will, for which the signer will be responsible.

Thus, with with overwhelmingly high probability, $H(PK||\cdot)$ avoids any "fixed points" of the adversary. Outside such fixed points, H is a random oracle. Let

$$H_{PK}(\cdot) \stackrel{def}{=} H(PK||\cdot).$$

Then H_{PK} is, with overwhelmingly high probability, a random oracle. Therefore, because the signature scheme was secure when implemented with a random oracle, it will be secure (with high probability) when implemented with H_{PK}.

5 Conclusions

Adversarial hashing is a real problem. Whether a digital signature scheme with a random oracle is secure assuming, say, that factoring or discrete logs are hard, may not be too relevant if we de-couple the design of hashing and signing. In fact, even if factoring or discrete logs were easy, rather than discovering such factoring or discrete log algorithms, an adversary who is given the chance of designing the hash function may be better off mounting a proper adversarial-hashing attack in order to forge signatures.

Of course, an adversary possessing a secret key enabling it to find collisions at will, will always defeat any hash-and-sign scheme. But, of course too, embedding such omni-powerful structures in a hash function without being detected may be too hard. We hope that our paper will encourage further study of the problem of adversarial hashing.

References

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communication Security*, November 1993. Revised version appears in `http://www-cse.ucsd.edu/users/mihir/papers/crypto-papers.html`.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In U. Maurer, editor, *Advances in Cryptology—Eurocrypt 96, Lecture Notes in Computer Science Vol. 1070*. Springer-Verlag, 1996. Revised version appears in `http://www-cse.ucsd.edu/users/mihir/papers/crypto-papers.html`.

[DBP96] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A strengthened version of RIPEMD. In D. Gollmann, editor, *Fast Software Encryption. Third International Workshop Proceedings*. Springer-Verlag, 1996.

[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.

[IEE97] *IEEE P1363/D1 (Draft Version 1): Standard Specifications for Public Key Cryptography.* Institute of Electrical and Electronic Engineers, Inc., December 1997. Available from `http://stdsbbs.ieee.org/groups/1363/`.

[NIS94] *FIPS Publication 186: Digital Signature Standard (DSS).* National Institute of Standards and Technology (NIST), May 1994. Available from `http://csrc.nist.gov/fips/`.

[NIS95] *FIPS Publication 180-1: Secure Hash Standard.* National Institute of Standards and Technology (NIST), April 1995. Available from `http://csrc.nist.gov/fips/`.

[NR96] K. Nyberg and R.A. Rueppel. Message recovery for signature schemes based on the discrete logarithm problem. *Designs, Codes and Cryptography*, 7(1–2):61–81, January 1996.

[Rab79] Michael O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical Report MIT/LCS/TR-212, Massachusetts Institute of Technology, Cambridge, MA, January 1979.

[Riv92] Ronald L. Rivest. *IETF RFC 1321: The MD5 Message-Digest Algorithm.* Internet Activities Board, April 1992. Available from `http://ds.internic.net/rfc/rfc1321.txt`.

[RSA93] *PKCS #1: RSA Encryption Standard. Version 1.5.* RSA Laboratories, November 1993. Available from `http://www.rsa.com/rsalabs/pubs/PKCS/`.

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

[Wil80] Hugh C. Williams. A modification of the RSA public-key encryption procedure. *IEEE Transactions on Information Theory*, IT-26(6):726–729, November 1980.