10 More on Signatures and the Public-Key Infrastructure

10.1 Random Oracle Model and Full-Domain-Hash

Very efficient stateless signatures seem to come from the so-called *random oracle model*, formally introduced by Bellare and Rogaway [BR93]. The idea is that often people use hash function such as SHA-1 [NIS95] as something that produces random-looking outputs. What if we really had a truly random function available to everyone (signer, verifier and adversary alike)?

In the random oracle model, the definition of signature scheme changes as follows: all three algorithms $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver})$ are now oracle algorithms $(\mathrm{Gen}^?, \mathrm{Sig}^?, \mathrm{Ver}^?)$; the adversary $E^?$ now has access to two oracles $E^{?,?}$. The new oracle will be random. Namely, the experiment changes as follows:

```
exp-forge(k)
0. Let $R : \{0,1\}^* \to \{0,1\}$ be a function chosen uniformly at random from all possible functions.
1. $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathrm{Gen}^R(1^k)$
2. $(m, \sigma) \leftarrow E^{\mathrm{Sig}^R_{\mathrm{SK}}(\cdot),R}(1^k, \mathrm{PK})$
3. If $m$ was not queried by $E$ to its signing oracle and $\mathrm{Ver}^R_{\mathrm{PK}}(m, \sigma) = 1$, output 1. Else output 0.
```

The rest of the definition stays the same. Note that the adversary has to be built obliviously to the oracle R, and work for a random choice of R.

In this model, we could build signature schemes more easily. Specifically, let (n, e) be an RSA public key, and (n, d) be the corresponding RSA secret key. Let $H : \{0,1\}^* \to \mathbb{Z}_n^*$ be a random function (it can be easily built out of $R : \{0,1\}^* \to \{0,1\}$). To sign m, compute $h = H(m)$, and $\sigma = h^d \bmod n$. To verify, compute $h = H(m)$ and check if it equals $\sigma^e \bmod n$.

More generally, let $\{f_i : D_i \to D_i\}$ be a trapdoor permutation family (such a family comes with the following probabilistic polynomial-time algorithms: the algorithm GenT to generate i and trapdoor t, an algorithm to compute $f_i(x)$ given i and $x \in D_i$, and an algorithm to compute $f_i^{-1}(y)$ given t and y). Let $H : \{0,1\}^* \to D_i$ denote the random oracle. Let Full Domain Hash (FDH) be the following signature scheme:

- Gen picks a trapdoor permutation: runs GenT to generate $\mathrm{PK} = i$ and $\mathrm{SK} = t$
- $\mathrm{Sig}(\mathrm{SK}, m)$ computes and outputs $s = f_i^{-1}(H(m))$
- $\mathrm{Ver}(\mathrm{PK}, m, s)$ checks if $f(s) = H(m)$

Theorem 1 ([BR93]). *Full Domain Hash is secure in the random oracle model.*

Proof. We will show security by reduction to the one-wayness of f. Indeed, suppose F is a forger for FDH. Then we will build an inverter *Inv* for f. Given an index i and a random value $y \in D_i$, *Inv* has to find $f_i^{-1}(y)$. Suppose F asks q_{hash} hash queries $a_1, \ldots, a_{q_{hash}}$ and q_{sig} signature queries $m_1, \ldots, m_{q_{sig}}$, and then outputs a forgery (m, s). Without loss of generality, assume that before m_j is queried to a signing oracle, it is queried to the hash oracle (if not, *Inv* can perform the query to the hash oracle itself before it answers the signing query). Same for the final forgery m: assume that before being output, it is queried to the hash oracle.

Inv proceeds as follows.

1. *Inv* chooses a random k between 1 and $q_{hash} + q_{sig} + 1$.
2. *Inv* sets $\mathrm{PK} = i$ and runs $F(\mathrm{PK})$.
3. Upon receiving a hash query a_j, if a_j has been queried before, then return the same answer as before. If not, then if $j = k$, return y, and store h_j. Else return choose a random $s_j \in D_i$ and return $h_j = f(s_j)$; store (h_j, s_j).

4. Upon receiving a signature query m_ℓ, find the hash query on the same message, $a_j = m_\ell$ (we assumed that such a query exists). If $j = k$, abort with failure. Else output s_j.

5. Upon receiving the forgery (m, s), find the hash query on m, $a_j = m$ (again, we assumed such a query exists). If $j = k$, output s. Else abort with failure.

Observe that if *Inv* does not abort with failure, then the view of F is exactly the same as in the real execution: F sees random values from D_i for hash values, and their correct unique inverses under f_i as signatures. Observe also that if *Inv* is lucky and guessed correctly the value k (i.e., guessed correctly which hash query the forgery would correspond to), then it will not abort with failure (because F is not allowed to ask signature queries on its eventual forgery m, so $m_\ell \neq m$ for any ℓ), and, moreover then s will be the value that *Inv* needs to output, i.e., $s = f_i^{-1}(H(m)) = f_i^{-1}(y)$. Thus, *Inv* will succeed with probability at least $\epsilon/(q_{hash} + q_{sig} + 1)$, where ϵ is F's success probability. Thus, if ϵ isn't negligible, then neither is the inversion probability and hence f is not one-way. □

10.2 Problems with Random Oracles

The problem with the random oracle model is that it doesn't really model real life. In real life, there are no random functions. There is usually a single fixed hash function that one uses, such as SHA-1. Therefore, even if the adversary may not work for a random function, it may work for this specific one, if designed with it in mind.

In fact, even if we think that the adversary was fixed before SHA-1 was designed, we still cannot say that SHA-1 was chosen at random. A simple counting argument shows this: the number of functions $\{0,1\}^k \to \{0,1\}$ is 2^{2^k}, hence a random function takes $\log_2 2^{2^k} = 2^k$ bits to represent. But this is an exponentially large number of bits, so most functions cannot even be represented.[1] The mere fact that hash functions are polynomial-time computable makes them "special"—it is theoretically possible that the adversary would fail for a random function, but not for a polynomial-time one.

In fact, Canneti, Goldreich and Halevi [CGH98] constructed an *artificial* counterexample: one that is provably secure in the random oracle model but insecure when the random oracle is instantiated in real-life with any polynomial-time computable function.

The status of the random oracle model, thus, is as follows: it allows us to "prove" a whole lot of practical signature schemes secure (including, in addition to Full-Domain-Hash, [FS86, Sch89, BR96] and others), as well as a lot of encryption schemes and other things, but the meaning of these proofs is uncertain (as opposed to proofs in the model without random oracles, which clearly imply that the scheme cannot be broken without violating the security assumption). It continues to be used because of its power, but it would be very nice if someone figured out how to prove these things without random oracles to give us more assurance that these are really secure. I personally view it as a way to acknowledge our failures: there are a lot of constructions that seem secure on some intuitive level, but we can't prove them secure in the standard model. So (hopefully until we have a real proof of security) we prove the secure in this funny fake model.

10.3 Signature Buzzwords

Here are some more buzzwords that we won't study in any detail for lack of time.

The most used signature scheme is probably PKCS #1 v. 1.5 [RSA93] (which is essentially the same as Full-Domain-Hash, except that the hash length is usually 160 bits, where as the length of n tends to be much larger; the remaining bits are filled-in with 1's; the result is not proven secure even in the random oracle model). It is getting replaced by version 2.1 [RSA02], which contains a Full-Domain-Hash-like

[1] For those who know a bit of set theory, a similar counting argument is even powerful when you consider functions $\{0,1\}^* \to \{0,1\}$. The number of such functions is uncountable (the same as the number of reals, $2^{\aleph_0}$), while the number of possible algorithms, or Turing machines, is countable (the same as the number of integers, $\aleph_0$.)

scheme of [BR96]. Some of the commonly mentioned schemes are DSA/DSS ("Digital Signature Algorithm/Standard"), standardized in [NIS94] (heuristically, but not provably, based on discrete logarithms); Fiat-Shamir [FS86] provably based on factoring in the random oracle model; Schnorr [Sch89] provably based on discrete logarithms in the random oracle model; and Guillou-Quisquater [GQ88] provably based on the RSA assumption in the random oracle model.

10.4 Use of Signatures and Public-Key Infrastructure

This is merely a short summary of the long discussion in class.

Signature tie messages to public keys, but they don't tie public-keys to their owners. For example, if you wish to get an authentic stock quote, it must be signed by the source, you must trust the source, and you must know the public key of the source.

This is just like a problem in the physical world, where we recognize one another by face or voice or handwriting instead of by key. We are usually introduced to one another by people we know. This leads to the idea of "certificate" (proposed by Kohnfelder [Koh78] in an undergraduate thesis), which is nothing more than a "letter of introduction" signed by someone you know. A certificate is a document signed by a certifying authority (CA) that says something like "the public key of www.bu.edu is x." You need to trust the CA (i.e., the humans who run the CA) to check that indeed someone authorized by "www.bu.edu" presents the public key for signature (otherwise, you end up with a fake certificate), and you also need to have the public-key of the CA (but that is usually built into your browser, since you need to trust your browser to verify signatures, anyway). By the way, there is a serious issue of how do you know your software, or the compiler with which it was compiled, or the compiler with which the compiler was compiled, etc., is trustworthy, which is not currently solved; see [Tho84] for an introduction to these issues.

Note that certificates need not be stored with the CA: each user can simply present the certificate together with a signature. Also note that CA is not trusted with any secrets beyond its own secret key. Nonetheless, it is trusted to verify identity, and must be able to revoke certificates in case of mistakes. Revocation, in particular, presents a serious issue, which we did not have time to address in detail; people often use on-line certificate status protocol (OCSP), which in some sense defeats the purpose of a certificate, because now the CA must be on-line; people also use certificate revocation lists (CRL), which are lists of revoked certificates published and signed by the CA. There are much more innovative approaches being proposed and marketed, based on Merkle trees, hash chains, and other fun things, but we don't have time to talk about them.

We then talked a bit about why we all use passwords instead of certificates. Partly it's because of historical reasons and entrenched mentality, and partly it's because of business reasons. Businesses don't want to trust other to verify identities of their clients. It's also much easier to explain the need for a password to a user than the need for a certificate (not mention the procedures needed to create and install a personal certificate in one's browser). However, passwords are making us all less secure, particularly because most of us use the same password on many sites (thus, a single untrustworthy site can pretend to be you and log in into any other site where you have an account with the same password).

We started asking "but how do we authenticate a user to a bank without a user certificate" and gave the example of encrypting a string that includes the password and username and instructions to transfer money to a friend. This doesn't work, because encryption doesn't provide authenticity and adversary could change the instructions to transfer money to his own account without changing the authentication. What you need here is so-called "non-malleable" encryption, which we haven't really studied (but FYI, the so-called OAEP scheme [**?**], which is now a PKCS #1 [RSA02] is non-malleable in the random oracle model based on the RSA assumption).

It would be much easier if each user had a certificate. However, banks for various reasons don't like to do that. Even here at BU we use a password-based authentication system known as Kerberos (which uses old symmetric techniques). We gave the example of a university using user certificates, and the advantages it gives: one can give a third party for example, a cable TV provider for the dorms, a proof that some people are university students without revealing the (private) list of all students. This is done as follows: when students need to sign up for cable service, they contact the cable company and sign their request using their certificate (i.e., using the secret key whose corresponding public key is certified by the university). They include the certificate with signature. The cable company only finds out about students who choose to contact it, and all the university needs to give the cable company is university's public key with which to verify certificates. Privacy of the students who are not signing up for cable service is not violated.

It would be nice to have a large-scale system for certifying users (a.k.a. Public-Key Infrastructure, or PKI), but the lack of unique names presents a serious problem. On a smaller scale, such as bank or a university, you can use account number or student id numbers, but what do you use on a worldwide scale?

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*, pages 62–73, November 1993. Revised version available from `http://www.cs.ucsd.edu/~mihir/`.

[BR96] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In Ueli Maurer, editor, *Advances in Cryptology—EUROCRYPT 96*, volume 1070 of *Lecture Notes in Computer Science*, pages 399–416. Springer-Verlag, 12–16 May 1996. Revised version appears in `http://www-cse.ucsd.edu/users/mihir/papers/crypto-papers.html`.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 209–218, Dallas, Texas, 23–26 May 1998.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology—CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987, 11–15 August 1986.

[GQ88] Louis Claude Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In Shafi Goldwasser, editor, *Advances in Cryptology—CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 216–231. Springer-Verlag, 1990, 21–25 August 1988.

[Koh78] Loren M. Kohnfelder. Towards a practical public-key cryptosystem. B.S. Thesis, supervised by L. Adleman, MIT, Cambridge, MA, May 1978.

[NIS94] FIPS publication 186: Digital signature standard (DSS), May 1994. Available from `http://csrc.nist.gov/fips/`.

[NIS95] FIPS publication 180-1: Secure hash standard, April 1995. Available from `http://csrc.nist.gov/fips/`.

[RSA93] PKCS #1: RSA encryption standard. Version 1.5, November 1993. Available from `http://www.rsaisecurity.com/rsalabs/pkcs/`.

[RSA02] PKCS #1: RSA encryption standard. Version 2.1, June 2002. Available from `http://www.rsaisecurity.com/rsalabs/pkcs/`.

[Sch89] C. P. Schnorr. Efficient identification and signatures for smart cards. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology—EUROCRYPT 89*, volume 434 of *Lecture Notes in Computer Science*, pages 688–689. Springer-Verlag, 1990, 10–13 April 1989.

[Tho84] Ken Thompson. Reflections on trusting trust. *Communications of the ACM*, 27(8):761–763, August 1984. Available from `http://www.acm.org/classics/sep95/`.