Algorithmic number theory, edited by Buhler and Stevenhagen, to appear.

PROTECTING COMMUNICATIONS AGAINST FORGERY

DANIEL J. BERNSTEIN

Abstract. This paper is an introduction to cryptography. It covers secret-key message authentication codes, unpredictable random functions, public-key secret-sharing systems, and public-key signature systems.

1. Introduction

Cryptography protects communications against espionage: an eavesdropper who intercepts a message will be unable to decipher it. This is useful for many types of information: credit-card transactions, medical records, love letters.

There is another side to cryptography. Cryptography protects communications against sabotage: a forger who fabricates or modifies a message will be unable to fool the receiver. This is useful for *all* types of information. If the receiver does not care about the authenticity of a message, why is he listening to the message in the first place?

This paper explains how cryptography prevents forgery. Section 2 explains how to protect n messages if the sender and receiver share $128(n+1)$ secret bits. Section 3 explains how the sender and receiver can generate many shared secret bits from a short shared secret. Section 4 explains how the sender and receiver can generate a short shared secret from a public conversation. Section 5 explains how the sender can protect a message sent to many receivers, without sharing any secrets.

2. Unbreakable secret-key authenticators

Here is a protocol for transmitting a message when the sender and receiver both know certain secrets:

$$\begin{tikzcd}
 & \text{Secrets } p,k \arrow[d] & & \text{Secrets } p,k \arrow[d] \\
\text{Message } m \arrow[r] & \begin{array}{c}\text{Authenticated}\\ \text{message } m,a\end{array} \arrow[r] & \begin{array}{c}\text{Possibly forged}\\ \text{message } m',a'\end{array} \arrow[r] & \text{Verification}
\end{tikzcd}$$

The message is a polynomial $m \in F[x]$ with $m(0) = 0$ and $\deg m \le 1000000$. Here F is the field $(\mathbf{Z}/2)[y]/(y^{128} + y^9 + y^7 + y^2 + 1)$ of size 2^{128}. The secrets are two independent uniform random elements p, k of F.

The sender transmits (m, a) where $a = m(p) + k$. The forger replaces (m, a) with (m', a'); if the forger is inactive then $(m', a') = (m, a)$. The receiver discards (m', a') unless $a' = m'(p) + k$.

The extra information a is called an **authenticator**.

Date: 2004.09.06. Permanent ID of this document: `9774ae5a1749a7b256cc923a7ef9d4dc`.
2000 *Mathematics Subject Classification.* Primary 94A62.

Security. I claim that the forger has chance smaller than 2^{-108} of fooling the receiver, i.e., of finding (m', a') with $m' \neq m$ and $a' = m'(p) + k$. The proof is easy. Fix (m, a) and (m', a'), and count pairs (p, k):

- There are exactly 2^{128} pairs (p, k) satisfying $a = m(p) + k$. Indeed, there is exactly one possible k for each possible p.
- Fewer than 2^{20} of these pairs also satisfy $a' = m'(p) + k$, if m' is different from m. Indeed, any qualifying p would have to be a root of the nonzero polynomial $m - m' - a + a'$; this polynomial has degree at most 1000000, so it has at most $1000000 < 2^{20}$ roots.

Thus the conditional probability that $a' = m'(p) + k$, given that $a = m(p) + k$, is smaller than $2^{20}/2^{128} = 2^{-108}$.

In practice, the receiver will continue listening for messages after discarding a forgery, so the forger can try again and again. Consider a persistent, wealthy, long-lived forger who tries nearly 2^{75} forgeries by flooding the receiver with one billion messages per second for one million years. His chance of success—his chance of producing at least one (m', a') with $a' = m'(p) + k$ and with m' not transmitted by the sender—is still smaller than $2^{-108}2^{75} = 2^{-33}$.

Handling many messages. One can use a single p with many k's to protect a series of messages:

$$\begin{tikzcd}
 & \begin{matrix}\text{Secrets}\\ p, k_1, k_2, \ldots\end{matrix} \arrow[d] & & \begin{matrix}\text{Secrets}\\ p, k_1, k_2, \ldots\end{matrix} \arrow[d] \\
\begin{matrix}\text{Message } m,\\ \text{number } n\end{matrix} \arrow[r] & \begin{matrix}\text{Authenticated}\\ \text{message } n, m, a\end{matrix} \arrow[r] & \begin{matrix}\text{Possibly forged}\\ \text{message } n', m', a'\end{matrix} \arrow[r] & \text{Verification}
\end{tikzcd}$$

The sender and receiver share secrets $p, k_1, k_2, k_3, \ldots$; as above, $(p, k_1, k_2, k_3, \ldots)$ is a uniform random sequence of elements of F. The sender transmits the nth message m as (n, m, a) where $a = m(p) + k_n$. The receiver discards (n', m', a') unless $a' = m'(p) + k_{n'}$.

In this context n is called a **nonce** and a is again called an **authenticator**. The random function $(n, m) \mapsto m(p) + k_n$ is called a **message authentication code** (MAC).

The forger's chance of success—his chance of producing at least one (n', m', a') with $a' = m'(p) + k_{n'}$ and with m' different from any message transmitted by the sender—is smaller than $2^{-108}D$, where D is the number of forgery attempts. This is true even if the forger sees all the messages transmitted by the sender. It is true even if the forger can influence the choice of those messages, perhaps responding dynamically to previous authenticators. In fact, is is true even if the forger has complete control over each message!

Define an **attack** as an algorithm that chooses a message m_1, sees the sender's authenticator $m_1(p) + k_1$, chooses a message m_2, sees the sender's authenticator $m_2(p) + k_2$, etc., and finally chooses (n', m', a'). Define the attack as **successful** if $a' = m'(p) + k_{n'}$ and $m' \notin \{m_1, m_2, \ldots\}$. Then the attack is successful with probability smaller than 2^{-108}. The proof is, as above, a simple matter of counting.

Of course, if the forger actually has the power to choose a message m_1 for the sender to authenticate, then the forger does not need to modify messages in transit.

Real senders restrict the messages $m_1, m_2, \dots$ that they authenticate, and thus restrict the possible set of attacks. But the security guarantee does not rely on any such restrictions.

History. Gilbert, MacWilliams, and Sloane in [17, section 9] introduced the first easy-to-compute unbreakable authenticator, using a long shared secret for a long message. Wegman and Carter in [38, section 3] proposed the form $h(m) + k_n$ for an authenticator and pointed out that a short secret could handle a long message.

There is now a huge literature on unbreakable MACs. For surveys see [29] and my paper [7]. For two state-of-the-art systems see [9] and [7].

3. Conjecturally unpredictable random functions

Here is a protocol that is *conjectured* to protect a series of messages:

$$\begin{tikzcd}
\text{Secret } s \arrow[r] & p, k_1, k_2, \dots \arrow[d] & \text{Secret } s \arrow[r] & p, k_1, k_2, \dots \arrow[d] \\
\begin{matrix}\text{Message } m,\\ \text{number } n\end{matrix} \arrow[r] & \begin{matrix}\text{Authenticated}\\ \text{message } n, m, a\end{matrix} \arrow[r] & \begin{matrix}\text{Possibly forged}\\ \text{message } n', m', a'\end{matrix} \arrow[r] & \text{Verification}
\end{tikzcd}$$

The sender and receiver share a secret uniform random 128-bit string s. The sender and receiver compute $p = \mathrm{SLASH}(0, s)$, $k_1 = \mathrm{SLASH}(1, s)$, $k_2 = \mathrm{SLASH}(2, s)$, etc. The sender transmits the nth message m as (n, m, a) where $a = m(p) + k_n$. The receiver discards (n', m', a') unless $a' = m'(p) + k_{n'}$.

The function SLASH (defined below) takes 512 bits of input. Message numbers n are assumed to be at most $2^{128} - 1$; a pair (n, s) is then encoded as a 512-bit input $(n_0, n_1, \dots, n_{127}, 0, 0, \dots, 0, s_0, s_1, \dots, s_{127})$ where $n = n_0 + 2n_1 + \cdots + 2^{127} n_{127}$. SLASH then produces 128 bits of output. The result has no apparent structure aside from its computability.

Note that the sender and receiver can compute $\mathrm{SLASH}(n, s)$ when they need it, rather than storing the long string $(p, k_1, k_2, \dots)$.

Security. A forger, given several authenticated messages, might try to solve for s. Presumably there is only one choice for s consistent with all the authenticators. However, the fastest *known* method of solving for s is to search through all 2^{128} possibilities. This is far beyond the computer power available today.

Is there a faster attack? Perhaps. We believe that this protocol is unbreakable, but we have no proof. (The random string $(p, k_1, k_2, \dots)$ is not uniform, so the proof in Section 2 does not apply.) On the other hand, this protocol has the advantage of using only 128 shared secret bits to handle any number of messages.

Unpredictability. Let u be a uniform random function from $\{0, 1, 2, \dots\}$ to F. Consider oracle algorithms A that print 0 or 1. What is the difference between

- the probability that A prints 1 using $n \mapsto \mathrm{SLASH}(n, s)$ as an oracle and
- the probability that A prints 1 using u as an oracle?

It is conjectured that the difference is smaller than 2^{-40} for every A that finishes in at most 2^{80} steps. In short, $n \mapsto \mathrm{SLASH}(n, s)$ is conjectured to be **unpredictable**.

If $n \mapsto \mathrm{SLASH}(n, s)$ is, in fact, unpredictable, then this authentication protocol is unbreakable: a fast algorithm that makes D forgery attempts cannot succeed with probability larger than $2^{-105} D + 2^{-40}$.

The SLASH **definition.** Say $x_0, x_1, \ldots, x_{15}$ are 32-bit strings. For $i \geq 16$ define $x_i = x_{i-16} + ((x_{i-1} + \delta_i) \oplus (x_{i-1} \lll 7))$. Then $\mathrm{SLASH}(x_0, x_1, \ldots, x_{15})$ is the 256-bit string $(x_0 \oplus x_{520}, x_1 \oplus x_{521}, \ldots, x_7 \oplus x_{527})$. In contexts where only 128 bits are required, the first 128 bits are used.

Notation: $(a_0, a_1, \ldots, a_{31}) + (b_0, b_1, \ldots, b_{31}) = (c_0, c_1, \ldots, c_{31})$ means that $a_0 + 2a_1 + \cdots + 2^{31}a_{31} + b_0 + 2b_1 + \cdots + 2^{31}b_{31} \equiv c_0 + 2c_1 + \cdots + 2^{31}c_{31} \pmod{2^{32}}$; $(a_0, a_1, \ldots, a_{31}) \oplus (b_0, b_1, \ldots, b_{31}) = (c_0, c_1, \ldots, c_{31})$ means that $a_i + b_i \equiv c_i \pmod 2$ for each i; $(a_0, a_1, \ldots, a_{31}) \lll 7 = (c_0, c_1, \ldots, c_{31})$ means that $c_7 = a_0$, $c_8 = a_1$, $\ldots$, $c_{31} = a_{24}$, $c_0 = a_{25}$, $\ldots$, $c_6 = a_{31}$; and δ_i means the string $(c_0, c_1, \ldots, c_{31})$ such that $c_0 + 2c_1 + \cdots + 2^{31}c_{31} \equiv 2654435769 \lfloor i/16 \rfloor \pmod{2^{32}}$.

History. Turing introduced the concept of unpredictability in [35]: "Suppose we could be sure of finding [laws of behaviour] if they existed. Then given a discrete-state machine it should certainly be possible to discover by observation sufficient about it to predict its future behaviour, and this within a reasonable time, say a thousand years. But this does not seem to be the case. I have set up on the Manchester computer a small programme using only 1000 units of storage, whereby the machine supplied with one sixteen figure number replies with another within two seconds. I would defy anyone to learn from these replies sufficient about the programme to be able to predict any replies to untried values."

The literature is full of very quickly computable short random functions that seem difficult to predict. Here **short** means that the random function is determined by a short uniform random string. See [33], [23], [28], and [27] for many examples. A typical example is more complicated than SLASH but somewhat faster.

Beware that the literature is also full of definitions that distract attention from unpredictability. For example, a **block cipher** is a short random inverse pair of functions (f, f^{-1}). One hopes that (f, f^{-1}) is indistinguishable from a uniform random inverse pair of functions. This indistinguishability implies unpredictability of f if the input size of f is large enough, say 256 bits; but the extra constraint of invertibility is unnecessary for applications and excludes many good designs.

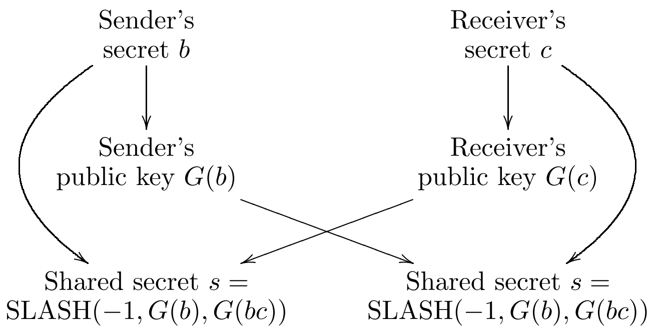
Blum, Blum, and Shub in [12] constructed a fast short random function with a small input, and proved that any fast algorithm to predict that function could be turned into a surprisingly fast algorithm to factor integers. Naor and Reingold in [26] constructed fast random functions with large inputs and with similar guarantees of unpredictability. These functions are never used in practice, because they are not nearly as fast as state-of-the-art block ciphers; but they show that unpredictability is not a silly concept.

Unpredictability has an interesting application to complexity theory: one can use it to turn fast probabilistic algorithms into reasonably fast deterministic algorithms. This was pointed out by Yao in [42]. It is now widely believed that the complexity classes BPP and P are identical, i.e., that everything decidable in polynomial time with the help of randomness is also decidable in polynomial time deterministically. See [18, Section 3.4].

The name "unpredictable" has several aliases in the literature. See my paper [6, Section 2] for further discussion.

4. Public-key secret sharing

Here is a protocol for the sender and receiver to generate a 128-bit shared secret from a public conversation:



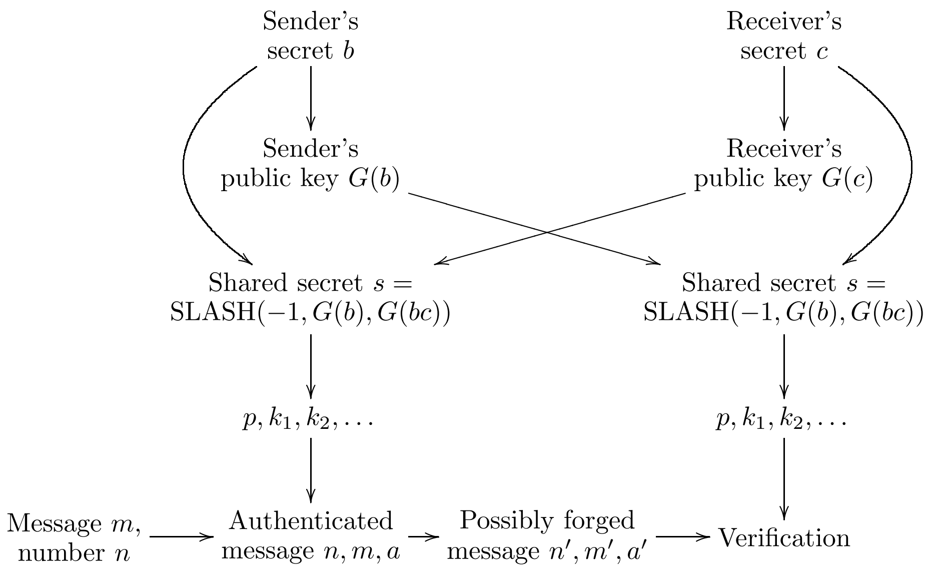
The sender starts from a secret uniform random $b \in 16\mathbf{Z}$ with $0 < b \leq 2^{225}$. The sender computes and announces a **public key** $G(b)$. Here $G(b)$ is, by definition, the x-coordinate of the bth multiples of the points $(53(2^{224}-1)/(2^8-1), \pm \dots)$ on the elliptic curve $y^2 = x^3 + 7530x^2 + x$ over the field $\mathbf{Z}/(2^{226}-5)$. It is not difficult to compute $G(b)$ from b; see [10], [19], and Chapter 3 of this book.

Similarly, the receiver starts from a secret uniform random $c \in 16\mathbf{Z}$ with $0 < c \leq 2^{225}$. The receiver computes and announces a public key $G(c)$.

The sender and receiver are assumed to receive correct copies of $G(b)$ and $G(c)$ from each other. Subsequent messages are protected against forgery, but the public keys themselves must be protected by something outside this protocol.

The sender now computes $G(bc)$; it is not difficult to compute $G(bc)$ from b and $G(c)$, both of which are known to the sender. The receiver computes $G(bc)$ from c and $G(b)$ in the same way. Finally, the sender and receiver both compute $s =$ SLASH$(-1, G(b), G(bc))$. Here $(-1, G(b), G(bc))$ is encoded as the 512-bit string $(g_0, g_1, \dots, g_{225}, 1, 1, 1, \dots, 1, h_0, h_1, \dots, h_{225})$ where $G(b) = g_0 + 2g_1 + \cdots + 2^{225}g_{225}$ and $G(bc) = h_0 + 2h_1 + \cdots + 2^{225}h_{225}$.

As in Section 3, the sender and receiver can use this shared secret s to protect the authenticity of a series of messages:



The sender can reuse his secret b with other receivers: given the public key $G(d)$ of another receiver, the sender computes $\mathrm{SLASH}(-1, G(b), G(bd))$ and continues as above. The sender and receiver can also reverse roles, using $\mathrm{SLASH}(-1, G(c), G(bc))$ and $\mathrm{SLASH}(-1, G(d), G(bd))$ for messages sent in the opposite direction.

Security. The complete definition of security here is more complicated than it was in Sections 2 and 3, because the forger has more power. In particular, the forger is given the public keys. The forger can also feed a number $G(c)$ to the sender (without necessarily knowing what c is) and receive authenticators computed using $\mathrm{SLASH}(-1, G(b), G(bc))$.

The fastest *known* attack is to start from the public key $G(b)$, perform about 2^{112} elliptic-curve operations, and deduce the secret b, after which the forger can compute $s = \mathrm{SLASH}(-1, G(b), G(bc))$ in the same way as the sender. As in Section 3, this is beyond the computer power available today, but there may be faster attacks.

Note that this attack does not depend on the details of SLASH. More precisely, consider a **generic protocol** in which the sender and receiver use an oracle for any 128-bit function in place of SLASH; then there is a **generic attack** in which the forger, having access to the same oracle, succeeds in forgeries after about 2^{112} elliptic-curve operations.

Any generic attack that succeeds for all 128-bit functions—or, more generally, succeeds with probability p on average over all 128-bit functions—can be converted into an algorithm at comparable speed that, given $G(b)$ and $G(c)$, computes $G(bc)$ with probability comparable to p. The idea of the proof is that if the algorithm never feeds $G(bc)$ to the oracle then it has no information about the shared secret. Of course, the value of this proof is limited, for two reasons: first, there might be faster non-generic attacks that exploit the structure of SLASH; second, we have no proof that computing $G(bc)$ from $G(b)$ and $G(c)$ is difficult.

History. Diffie and Hellman in [13] introduced the general idea of sharing a secret through a public channel. They also introduced the specific approach of exchanging public keys $2^b \bmod \ell$ and $2^c \bmod \ell$ to share a secret $2^{bc} \bmod \ell$; here ℓ is a fixed prime. The problem of computing $2^{bc} \bmod \ell$ from $(2^b \bmod \ell, 2^c \bmod \ell)$ is called the **Diffie-Hellman problem**.

There are surprisingly fast techniques to compute b from $2^b \bmod \ell$. See Chapters 8 and 9 of this book. Consequently one must choose a rather large prime ℓ in the Diffie-Hellman system.

Miller in [24], and independently Koblitz in [20], suggested replacing the unit group $(\mathbf{Z}/\ell)^*$ with an elliptic curve over $\mathbf{Z}/\ell$. No surprisingly fast techniques are known for the "elliptic-curve Diffie-Hellman problem" for most curves with near-prime order, so we *believe* that a relatively small value of ℓ, such as $\ell = 2^{226} - 5$, is safe. My elliptic curve $y^2 = x^3 + 7530x^2 + x$ over the field $\mathbf{Z}/(2^{226} - 5)$ has order $(2^{226} - 5) + 1 - 12000403261375786655687951397247436$, which is 16 times a prime.

Elliptic-curve computations involve more effort than unit-group operations, but this increase is outweighed by the reduction in the size of ℓ, so the Miller-Koblitz elliptic-curve variant is faster than the original Diffie-Hellman system. It also has shorter keys. The variant is becoming increasingly popular.

Fiat and Shamir in [16] proved that a generic attack on one protocol could be converted into an algorithm to solve an easy-to-state mathematical problem.

Bellare and Rogaway in [5] expanded the idea to more protocols. Many such proofs have now been published. See [21] for an exposition.

5. Public-key signatures

Here is a protocol—with no shared secrets—for the sender to protect messages sent to many receivers:

$$\begin{tikzcd}
 & \begin{array}{c}\text{Sender's}\\ \text{secret } p,q,z\end{array} \arrow[rr] \arrow[d] & & \begin{array}{c}\text{Sender's}\\ \text{public key } pq\end{array} \arrow[d] \\
\text{Message } m \arrow[r] & \begin{array}{c}\text{Signed}\\ \text{message}\\ m,e,f,h,r,s,t\end{array} \arrow[r] & \begin{array}{c}\text{Possibly forged}\\ \text{message}\\ m',e',f',h',r',s',t'\end{array} \arrow[r] & \text{Verification}
\end{tikzcd}$$

The sender starts from a secret uniform random 256-bit string z, and secret uniform random primes p, q in the interval $[2^{768}, 2^{768}+2^{766}]$ with $p \bmod 8 = 3$ and $q \bmod 8 = 7$; primality can be tested quickly, as explained in Chapter 6 of this book. The sender computes and announces the product pq, which is assumed to be transmitted correctly to all receivers. Subsequent messages are protected against forgery as follows.

Given a message m, the sender computes

- $r = \mathrm{SLASH}(-2, z, m) \bmod 16$;
- $h = H(r, m)$ where $H(r, m) = \mathrm{SLASH}(-12, r, m) + 2^{128}\,\mathrm{SLASH}(-13, r, m) + \cdots + 2^{1408}\,\mathrm{SLASH}(-23, r, m) + 1$;
- $u = h^{(q+1)/4} \bmod q$;
- $e = 1$ if $u^2 \equiv h \pmod q$, else $e = -1$;
- $v = (eh)^{(p+1)/4} \bmod p$;
- $f = 1$ if $v^2 \equiv eh \pmod p$, else $f = 2$;
- $w = f^{(3q-5)/4} u \bmod q$;
- $x = f^{(3p-5)/4} v \bmod p$;
- $y = w + q(q^{p-2}(x - w) \bmod p)$;
- $s = \min\{y, pq - y\}$; and
- $t = (fs^2 - eh)/pq$.

The sender then transmits (m, e, f, h, r, s, t).

At this point (e, f, h, r, s, t) is a **signature** of m under the public key pq. This means, by definition, that $e \in \{1, -1\}$; $f \in \{1, 2\}$; $r \in \{0, 1, \ldots, 15\}$; s and t are in $\{0, 1, \ldots, 2^{1536} - 1\}$; $h = H(r, m)$; and $fs^2 = tpq + eh$.

The receiver discards $(m', e', f', h', r', s', t')$ if (e', f', h', r', s', t') is not a signature of m'. The receiver can save time here by checking the equation $f'(s')^2 = t'pq + e'h'$ modulo a secret 128-bit prime.

Observe that signatures are different from authenticators: a signature can be verified by anyone, while an authenticator can be verified only by people who could have created the authenticator. The receiver can convince third parties that the sender signed a message; the receiver cannot convince third parties that the sender authenticated a message. Signatures are appropriate for public communications; authenticators are appropriate for private communications.

Security. Like the protocols in Sections 3 and 4, this protocol *appears* to make forgeries extremely difficult, even if the forger can inspect signatures on messages under his control. There are surprisingly fast techniques to factor pq into p, q—see Chapters 10 and 11 of this book—but for large pq these computations are beyond the computer power available today.

One can prove that any generic attack against this protocol can be converted into an algorithm at comparable speed to factor pq with comparable success probability. However, as in Section 4, the value of this proof is limited: there might be faster non-generic attacks, and we have no proof that factorization is difficult.

Message length. The above description of signatures presumes that $(-2, z, m)$ and $(-12, r, m)$ and so on are encoded as 512-bit strings to be fed to SLASH. Thus messages m must be very short.

One can handle longer messages by modifying SLASH to allow larger inputs. One can, for example, define $\mathrm{SLASH}(x_0, x_1, x_2, x_3)$, where each x_i is a 256-bit string, as $\mathrm{SLASH}(\mathrm{SLASH}(\mathrm{SLASH}(\mathrm{SLASH}(0, 0, x_0), 1, x_1), 2, x_2), 3, x_3)$.

History. Diffie and Hellman in [13] introduced the idea of public-key signatures. Rivest, Shamir, and Adleman in [32] are often credited with the first useful example; but the original RSA system is obviously breakable.

(In the original RSA system, s is a signature of m under a public key (n, e) if $s^e \equiv m \pmod{n}$. First obvious attack: the forger immediately computes the message $2^e \bmod n$ with signature 2. Second obvious attack: starting from m, the forger obtains from the sender a signature on the message $2^e m \bmod n$, and then divides the result by 2 modulo n.)

Rabin in [31] introduced the first useful signature system. The system shown above is Rabin's system with improvements by Williams in [40], Barwood, Wigley, and me. See my paper [8, Section 2] for a survey and comparison of Rabin-type systems. Recent results of Bleichenbacher, Coppersmith, and Gentry show that signatures and public keys can be compressed to a surprising extent.

There are many "cryptographic hash functions" that can be used in place of H; see, e.g., [23, Sections 9.3–9.4]. On the other hand, some hash functions have been broken; see, e.g., [36]. I offer \$1000 to the first person to publish a SLASH input whose output is 128 all-zero bits, or two different 512-bit SLASH inputs with the same 256-bit output.

There are other signature systems. One interesting example is the ElGamal system in [15], which uses Diffie-Hellman public keys. Keys and signatures in elliptic-curve variants of ElGamal's system are smaller than keys and signatures in Rabin-type systems; on the other hand, signature verification is slower. Rabin-type systems and ElGamal-type systems are both widely used.

References

[1] —, *20th annual symposium on foundations of computer science*, IEEE Computer Society, New York, 1979. MR 82a:68004.

[2] —, *23rd annual symposium on foundations of computer science*, IEEE Computer Society, New York, 1982. MR 85k:68007.

[3] —, *38th annual symposium on foundations of computer science*, IEEE Computer Society Press, Los Alamitos, 1997. ISBN 0–8186–8197–7.

[4] Victoria Ashby (editor), *First ACM conference on computer and communications security*, Association for Computing Machinery, New York, 1993.

[5] Mihir Bellare, Phillip Rogaway, *Random oracles are practical: a paradigm for designing efficient protocols*, in [4] (1993), 62–73.

[6] Daniel J. Bernstein, *How to stretch random functions: the security of protected counter sums*, Journal of Cryptology **12** (1999), 185–192. ISSN 0933–2790. URL: `http://cr.yp.to/papers.html`.

[7] Daniel J. Bernstein, *Floating-point arithmetic and message authentication*, to be incorporated into author's *High-speed cryptography* book. URL: `http://cr.yp.to/papers.html`.

[8] Daniel J. Bernstein, *Proving tight security for standard Rabin-Williams signatures*, to be incorporated into author's *High-speed cryptography* book. URL: `http://cr.yp.to/papers.html#rwtight`. ID `c30057d690a8fb42af6a5172b5da9006`.

[9] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, Phillip Rogaway, *UMAC: fast and secure message authentication*, in [39] (1999), 216–233. URL: `http://www.cs.ucdavis.edu/~rogaway/umac/`.

[10] Ian F. Blake, Gadiel Seroussi, Nigel P. Smart, *Elliptic curves in cryptography*, Cambridge University Press, Cambridge, 2000. ISBN 0–521–65374–6. MR 1 771 549.

[11] G. R. Blakley, David Chaum (editors), *Advances in cryptology: CRYPTO '84*, Lecture Notes in Computer Science, 196, Springer-Verlag, Berlin, 1985. ISBN 3–540–15658–5. MR 86j:94003.

[12] Lenore Blum, Manuel Blum, Michael Shub, *A simple unpredictable pseudo-random number generator*, SIAM Journal on Computing **15** (1986), 364–383. ISSN 0097–5397. MR 87k:65007. URL: `http://cr.yp.to/bib/entries.html#1986/blum`.

[13] Whitfield Diffie, Martin Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **22** (1976), 644–654. ISSN 0018–9448. MR 55:10141.

[14] Taher ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, in [11] (1985), 10–18; see also newer version [15]. MR 87b:94037.

[15] Taher ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory **31** (1985), 469–472; see also older version [14]. ISSN 0018–9448. MR 86j:94045.

[16] Amos Fiat, Adi Shamir, *How to prove yourself: practical solutions to identification and signature problems*, in [30] (1987), 186–194. MR 88m:94023.

[17] Edgar N. Gilbert, F. Jessie MacWilliams, Neil J. A. Sloane, *Codes which detect deception*, Bell System Technical Journal **53** (1974), 405–424. ISSN 0005–8580. MR 55:5306.

[18] Oded Goldreich, *Modern cryptography, probabilistic proofs and pseudorandomness*, Springer-Verlag, Berlin, 1999. ISBN 3–540–64766–X. MR 2000f:94029.

[19] Darrel Hankerson, Alfred Menezes, Scott Vanstone, *Guide to elliptic curve cryptography*, Springer, New York, 2004. ISBN 0–387–95273–X. MR 2054891.

[20] Neal Koblitz, *Elliptic curve cryptosystems*, Mathematics of Computation **48** (1987), 203–209. ISSN 0025–5718. MR 88b:94017.

[21] Neal Koblitz, Alfred J. Menezes, *Another look at "provable security"* (2004). URL: `http://www.cacr.math.uwaterloo.ca/~ajmeneze/publications/provable.pdf`.

[22] Hugo Krawczyk (editor), *Advances in cryptology: CRYPTO '98*, Lecture Notes in Computer Science, 1462, Springer-Verlag, Berlin, 1998. ISBN 3–540–64892–5. MR 99i:94059.

[23] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, *Handbook of applied cryptography*, CRC Press, Boca Raton, Florida, 1996. ISBN 0–8493–8523–7. MR 99g:94015. URL: `http://cacr.math.uwaterloo.ca/hac`.

[24] Victor S. Miller, *Use of elliptic curves in cryptography*, in [41] (1986), 417–426. MR 88b:68040.

[25] Gary L. Mullen, Peter Jau-Shyong Shiue (editors), *Finite fields: theory, applications, and algorithms*, American Mathematical Society, Providence, 1994. ISBN 0–8218–5183–7. MR 95c:11002.

[26] Moni Naor, Omer Reingold, *Number-theoretic constructions of efficient pseudo-random functions*, in [3] (1997), 458–467. URL: `http://www.wisdom.weizmann.ac.il/~naor/onpub.html`.

[27] James Nechvatal, Elaine Barker, Lawrence Bassham, William Burr, Morris Dworkin, James Foti, Edward Roback, *Report on the development of the Advanced Encryption Standard (AES)*, Journal of Research of the National Institute of Standards and Technology **106** (2001). URL: `http://nvl.nist.gov/pub/nistpubs/jres/106/3/cnt106-3.htm`.

[28] James Nechvatal, Elaine Barker, Donna Dodson, Morris Dworkin, James Foti, Edward Roback, *Status report on the first round of the development of the Advanced Encryption Standard*, Journal of Research of the National Institute of Standards and Technology **104** (1999). URL: `http://nvl.nist.gov/pub/nistpubs/jres/104/5/cnt104-5.htm`.

[29] Wim Nevelsteen, Bart Preneel, *Software performance of universal hash functions*, in [34] (1999), 24–41.

[30] Andrew M. Odlyzko (editor), *Advances in cryptology—CRYPTO '86: proceedings of the conference on the theory and applications of cryptographic techniques held at the University of California, Santa Barbara, Calif., August 11–15, 1986*, Lecture Notes in Computer Science, 263, Springer-Verlag, Berlin, 1987. ISBN 3–540–18047–8. MR 88h:94004.

[31] Michael O. Rabin, *Digitalized signatures and public-key functions as intractable as factorization*, Technical Report 212, MIT Laboratory for Computer Science, 1979. URL: `http://ncstrl.mit.edu/Dienst/UI/2.0/Describe/ncstrl.mit_lcs/MIT/LCS/TR-212`.

[32] Ronald L. Rivest, Adi Shamir, Leonard M. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM **21** (1978), 120–126. ISSN 0001–0782. URL: `http://cr.yp.to/bib/entries.html#1978/rivest`.

[33] Bruce Schneier, *Applied cryptography: protocols, algorithms, and source code in C*, 2nd edition, Wiley, New York, 1996. ISBN 0–471–12845–7.

[34] Jacques Stern (editor), *Advances in cryptology: EUROCRYPT '99*, Lecture Notes in Computer Science, 1592, Springer-Verlag, Berlin, 1999. ISBN 3–540–65889–0. MR 2000i:94001.

[35] Alan M. Turing, *Computing machinery and intelligence*, MIND **59** (1950), 433–460. ISSN 0026–4423. MR 12,208c.

[36] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, Hongbo Yu, *Collisions for hash functions MD4, MD5, HAVAL–128 and RIPEMD* (2004). URL: `http://eprint.iacr.org/2004/199/`.

[37] Mark N. Wegman, J. Lawrence Carter, *New classes and applications of hash functions*, in [1] (1979), 175–182; see also newer version [38]. URL: `http://cr.yp.to/bib/entries.html#1979/wegman`.

[38] Mark N. Wegman, J. Lawrence Carter, *New hash functions and their use in authentication and set equality*, Journal of Computer and System Sciences **22** (1981), 265–279; see also older version [37]. ISSN 0022–0000. MR 82i:68017. URL: `http://cr.yp.to/bib/entries.html#1981/wegman`.

[39] Michael Wiener (editor), *Advances in cryptology—CRYPTO '99*, Lecture Notes in Computer Science, 1666, Springer-Verlag, Berlin, 1999. ISBN 3–5540–66347–9. MR 2000h:94003.

[40] Hugh C. Williams, *A modification of the RSA public-key encryption procedure*, IEEE Transactions on Information Theory **26** (1980), 726–729. ISSN 0018–9448. URL: `http://cr.yp.to/bib/entries.html#1980/williams`.

[41] Hugh C. Williams (editor), *Advances in cryptology: CRYPTO '85*, Lecture Notes in Computer Science, 218, Springer, Berlin, 1986. ISBN 3–540–16463–4.

[42] Andrew C. Yao, *Theory and applications of trapdoor functions*, in [2] (1982), 80–91.

Department of Mathematics, Statistics, and Computer Science (M/C 249), The University of Illinois at Chicago, Chicago, IL 60607–7045

E-mail address: `djb@cr.yp.to`