CPSC/PMAT 669

Data Integrity and Digital Signatures

Mike Jacobson

Department of Computer Science
University of Calgary

Topic 8

Outline

Hash Function

Often referred to as the "work horse" of cryptography — they are ubiquitous in crypto.

Definition 1 (Hash function)

A function $H : \{0,1\}^* \to \{0,1\}^m$ ($m \in \mathbb{N}$) that is easy to compute. An image $x = H(M)$ is referred to as a *message digest* or a *digital fingerprint* or a *checksum* or simply a *hash*.

Hash functions thus satisfy two properties:

- *Compression*: H maps an input M of arbitrary bit length to an output of fixed bit length.
- *Ease of computation*: for any input M, $H(M)$ is easy to compute.

Cryptographic Requirements

Desirable properties for hash functions in the context of cryptography:

- *Pre-image resistance*: given any hash value x, it is computationally infeasible to find *any* input M for which $H(M) = x$.
- *Second pre-image resistance* or *weak collision resistance*: given any M, it is computationally infeasible to find $M' \neq M$ with $H(M) = H(M')$.
- *Collision resistance* or *strong collision resistance*: it is computationally infeasible to find two distinct inputs M and M' such that $H(M) = H(M')$.

Note that collision resistance is the strongest of these three requirements. In other words: collision resistance $\Rightarrow$ weak collision resistance $\Rightarrow$ pre-image resistance

Uses of Cryptographically-Secure Hash Functions

Definition 2

A hash function is *cryptographic(ally secure)* if it is collision resistant.

Some example applications:

- In digital signatures to prevent impersonation (sign $H(M)$ instead of M — later)
- Data integrity without secrecy (*e.g.* downloading large files, compare checksum before and after download)
- Data integrity with secrecy (see below)
- Commitment (can verify $H(M)$ to see if M was committed to)
- Randomness (*e.g.* one-time passwords, OAEP)

Eg. Data Integrity with Secrecy

Using hashing plus encryption:

- Sender sends $C = E_K(M \| x)$ with $x = H(M)$
- Receiver decrypts C to obtain M', x' and checks that $H(M') = x'$.

Idea:

- Adversary cannot manipulate ciphertext blocks in such a way that $H(M') = x'$.
- May be possible if H is not cryptographically secure (eg. WEP: combination of stream cipher and checksum).

Attacks on Hash Functions

Objectives of adversaries vs. hash functions:

- Find a pre-image: given any hash, create a corresponding message with that hash.
- Find a weak collision: given a message, modify it to another message with the same hash.
- Find a collision: find two messages with the same hash.

Brute-force Attacks

Like block ciphers, brute force should be the best attack.

For an m-bit hash function:

- Pre-images and weak collisions: 2^m attempts on average
- Strong collisions: $2^{m/2}$ attempts on average due to the *birthday paradox* — probability of having at least one duplicate out of k random numbers between 1 and n is of order $\sqrt{n}$

Recommended sizes: $m = 160$, 256, 394, 512 (provide 80, 128, 192, and 256 bits of security)

Cryptanalytic Attacks

Iterated hash functions are composed of rounds (most common design)

- Repeated use of *compression function* f — takes n-bit input from the previous step (chaining variable) and a b-bit block from M; produces n-bit output.
- Input to H : message M consisting of L b-bit blocks $Y_0, \ldots, Y_{L-1}$ (padded to suitable length).
- $CV_0 = IV =$ initial n-bit value (e.g. all zeros).
- $CV_i = f(CV_{i-1}, Y_{i-1})$, $1 \leq i \leq L$
- $H(M) = CV_L$

Iterated hash functions can be set up in such a way so that if f is collision-resistant, so is H (Merkle 1989 and Damgard 1989).

Idea for Attacking

Exploit the structure of the hash function (similar to block ciphers):

- Analytically attack the rounds of a hash function
- Focus on collisions in function f.
- Almost all widely-used hash function have succumbed to this type of attack (due to Wang et al).

SHA-1

Secure Hash Algorithm 1: developed by NIST in 1993 (FIPS 180 and FIPS 180-1).

- Iterated round hash function with hash length 160 bits

Finding collisions:

- Wang, Yin, Yu (Feb. 2005) — 2^{69} hash ops
- Wang, Yao, Yao (Aug. 2005) — 2^{63} hash ops
- Stephens (2012) — 2^{60} hash ops

Significantly less than theoretical maximum (2^{80}) — therefore, considered vulnerable.

Other Hash Functions

MD5 — 128-bit hash length, developed by Rivest.

- Essentially broken (Wang et. al., 2004). Can find MD5 collisions on a laptop in 8 hours or less (Klima, 2005).

Revised hash standard SHA-2 consisting of SHA-256, SHA-384, SHA-512

- modifications of SHA-1 to provide 128, 192, and 256 bits of security for compatibility with AES (see FIPS 180-4).
- current recommendation: use one of these in place of SHA-1.

Charles, Goren, Lauter (2009) — hash function based on expander graphs

- provable security: finding collisions reduces to computing computing isogenies between supersingular elliptic curves

See NIST's hash function page in the Cryptographic Tool Kit for more.

SHA-3: Keccak

After the 2005 attack on SHA-1, NIST initiated a competition for new hash algorithms, similar to the AES competition; see `//csrc.nist.gov/groups/ST/hash/`.

The SHA-3 winner, Keccak (pronounced "ketchuk") was announced on October 2, 2012.

- Invented by Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles van Assche.

Resources:

- NIST FIPS 202
- `http://keccak.noekeon.org/Keccak-reference-3.0.pdf`
- KECCAK presentation given to NIST by the Keccak inventors on Feb. 6, 2013 (on "handouts" page)

Sponge Construction

Keccak is based on a *sponge* design; see http://sponge.noekeon.org/.

- Hash function: arbitrary input length, fixed output length
- Stream cipher: fixed input length, arbitrary output length
- Sponge function: arbitrary input length, variable user-supplied output length

Sponges can be used to build various cryptographic primitives (stream ciphers, hash functions, message authentication codes)

Sponges – Overview

Ingredients of a sponge function:

- A *width* b (an integer)
- A *bit rate* r (an integer $< b$)
- An input S (a bit string of length b)
- A fixed-length permutation f that operates on S
- A padding rule "*pad*" that pads blocks of length r to blocks of length b.

The *capacity* of the sponge is the padding amount $c = b - r$.

The padding rule for Keccak simply appends the string $1\underbrace{00\cdots0}_{c\text{-2 zeros}}1$ to each r-bit block (called *multi-rate padding*).

Sponge Function – Absorb

The input to the *absorption* phase is the message M — padded so the total length is a multiple of r — consisting of r-bit blocks $P_1, \ldots, P_L$.

Absorption Phase — "x-or & permute"

$S \leftarrow 0^b$ (b zeros)
For $i = 1$ to L do
  $S \leftarrow S \oplus pad(P_i)$
  $S \leftarrow f(S)$
end for

Sponge Function – Squeeze

The *squeezing phase* outputs a hash of the message M whose bit length is a user-supplied value m.

Squeezing Phase — "append & permute"

$Z \leftarrow$ first r bits of S
While $length(Z) < m$ do
  $S \leftarrow f(S)$
  append the first r bits of S to Z
end while
$H(M) \leftarrow$ first m bits of Z

SHA-3 Specification

SHA-3/Keccak specifies

- hash lengths $m = 224, 256, 384, 512$ (just like SHA-2)
- capacities $c = 2m$
- widths $b = 25$, 50, 100, 200, 400, 800, 1600 (default is 1600)

The internal state to the Keccak permutation f , denoted A, is a 3-dimensional bit-array of dimensions $5 \times 5 \times 2^\ell$ where $0 \leq \ell \leq 6$, yielding the above widths (default is $\ell = 6$, with a state of dimensions $5 \times 5 \times 64$).

The Keccak permutation f iterates over multiple rounds. In SHA-3, the number of rounds N_r is $12 + 2\ell$. (E.g. $N_r = 24$ for for $b = 1600$.) Each round of f operates on the state A and is the composition of 5 functions:

$$\iota \circ \chi \circ \pi \circ \rho \circ \theta$$

where θ, ρ, π and χ are identical for each round, and ι incorporates *round constants* that vary by round.

The Keccak Permutation f

Input: bit string S of length b

Output: bit string S of length b

1. Convert S to a $5 \times 5 \times 2^\ell$ state A (where $b = 5 \cdot 5 \cdot 2^\ell$)
2. For $i = 0$ to $N_r - 1$ do
   $$A \leftarrow \iota(\chi(\pi(\rho(\theta(A)))), i)$$
3. Convert A to a string S of length b
4. Output S

The mathematical description of each of the 5 maps θ, ρ, π, χ and ι can be found on page 8 of *Keccak-reference-3.0.pdf*. They can all be implemented using only bitwise XOR, AND, NOT, but no table look-ups, arithmetic or data-dependent rotations (very fast).

Geography of Keccak States

State entries are denoted $A[x, y, z]$ where

$$0 \leq x \leq 4\ , \quad 0 \leq y \leq 4\ , \quad 0 \leq z \leq 2^\ell - 1\ .$$

E.g. for $b = 1600$ ($\ell = 6$), we have $0 \leq x \leq 4$, $0 \leq y \leq 4$, $0 \leq z \leq 63$.

Navigating States:

Rows:	$A[0, y, z]$ $A[1, y, z]$ $A[2, y, z]$ $A[3, y, z]$ $A[4, y, z]$
Columns:	$A[x, 0, z]$ $A[x, 1, z]$ $A[x, 2, z]$ $A[x, 3, z]$ $A[x, 4, z]$
Lanes:	$A[x, y, 0]$ $A[x, y, 1]$ $A[x, y, 2]$ $\cdots$ $A[x, y, 2^\ell - 1]$

Converting Bit Strings to States

Suppose the input string consists of bits

$$s_0, s_1, \ldots, s_{b-1} \ .$$

Then

$$A[x, y, z] = s_{2^\ell(5y+x)+z} \ .$$

So A is populated lane-wise, "floor" by "floor":

- starting with the bottom row of lanes (ground floor)
- followed by the row of lanes second from the bottom (second floor)
- followed by the middle, then the second from the top, then the top row of lanes

Converting Bit Strings to States (cont'd)

We assign the bits s_i $(0 \leq i \leq b-1)$ to A in the following order:

$y = 0$	$x = 0$	$z = 0, 1, \ldots 2^\ell - 1$
	$x = 1$	$z = 0, 1, \ldots 2^\ell - 1$
	$\vdots$	$\vdots$
	$x = 4$	$z = 0, 1, \ldots 2^\ell - 1$
$y = 1$	$x = 0$	$z = 0, 1, \ldots 2^\ell - 1$
	$x = 1$	$z = 0, 1, \ldots 2^\ell - 1$
	$\vdots$	$\vdots$
	$x = 4$	$z = 0, 1, \ldots 2^\ell - 1$
$\vdots$	$\vdots$	$\vdots$
$y = 4$	$x = 0$	$z = 0, 1, \ldots 2^\ell - 1$
	$\vdots$	$\vdots$
	$x = 4$	$z = 0, 1, \ldots 2^\ell - 1$

Converting States to Bit Strings

Conversion from the final state A to the bit string S is done in by reversing this process (order lane–row–column):

$$\begin{aligned} S = & \ A[0,0,0] \ A[0,0,1] \ \ldots \ A[0,0,2^\ell-1] \\ & A[1,0,0] \ A[1,0,1] \ \ldots \ A[1,0,2^\ell-1] \\ & A[2,0,0] \ A[2,0,1] \ \ldots \ A[2,0,2^\ell-1] \\ & A[3,0,0] \ A[3,0,1] \ \ldots \ A[3,0,2^\ell-1] \\ & A[4,0,0] \ A[4,0,1] \ \ldots \ A[4,0,2^\ell-1] \\ & A[0,1,0] \ A[0,1,1] \ \ldots \ A[0,1,2^\ell-1] \\ & \quad \ldots \\ & A[4,1,0] \ A[4,1,1] \ \ldots \ A[4,1,2^\ell-1] \\ & \quad \ldots \\ & A[0,4,0] \ A[0,4,1] \ \ldots \ A[0,4,2^\ell-1] \\ & \quad \ldots \\ & A[4,4,0] \ A[4,4,1] \ \ldots \ A[4,4,2^\ell-1] \end{aligned}$$

The Map θ

θ adds to each bit $A[x, y, z]$ the bitwise x-or of the parities of the two columns $A[x-1, *, z]$ and $A[x+1, *, z-1]$, where the x-index is taken modulo 5 and the z-index modulo 2^ℓ.

1. For all pairs (x, z) with $0 \leq x \leq 4$ and $0 \leq z \leq 2^{\ell-1}$ do
   // x-or all columns $A[x, *, z]$ to compute parities
   $C[x, z] \leftarrow A[x, 0, z] \oplus A[x, 1, z] \oplus A[x, 2, z] \oplus A[x, 3, z] \oplus A[x, 4, z]$
2. For all pairs (x, z) with $0 \leq x \leq 4$ and $0 \leq z \leq 2^{\ell-1}$ do
   $D[x, z] \leftarrow C[(x-1) \bmod 5, z] \oplus C[(x+1) \bmod 5, (z-1) \bmod 2^\ell]$
3. For all triples (x, y, z) with $0 \leq x \leq 4$, $0 \leq y \leq 4$, $0 \leq z \leq 2^{\ell-1}$ do
   $A[x, y, z] \leftarrow A[x, y, z] \oplus D[x, z]$

θ provides a high level of diffusion.

The Map ρ

ρ rotates the bits of each lane by adding to the z-coordinate an *offset* modulo 2^ℓ (circular shift along the lane) as given in the following table:

	$x=3$	$x=4$	$x=0$	$x=1$	$x=2$
$y=2$	153	231	3	10	171
$y=1$	55	276	36	300	6
$y=0$	28	91	0	1	190
$y=4$	120	78	210	66	253
$y=3$	21	136	105	45	15

Consult pages 12-13 of FIPS 202 or page 8 of *Keccak-reference-3.0.pdf* to see how these offsets are calculated.

ρ disperses *slices* $A[x, y, *]$ for more diffusion.

The Map π

π rearranges all the lanes, moving lane

$$A[x, y, *]$$

to lane

$$A[(x + 3y) \bmod 5, x, *] \ .$$

This lane dispersion provides yet more diffusion.

The Map χ

χ x-or's each bit $A[x, y, z]$ with the non-linear function of two bits in the same row given by

$$\overline{A}[(x + 1) \bmod 5, y, z] \ \wedge \ A[(x + 2) \bmod 5, y, z]$$

where $\overline{A}$ denotes the bit complement of A and $\wedge$ denotes logical "and" (multiplication modulo 2).

χ is the only non-linear map within Keccak.

The Map ι

ι x-or's the ℓ bits $A[0, 0, 2^j - 1]$ $(0 \leq j \leq \ell)$ with *round constants* $rc(j + 7i)$ where i is the round number.

Here, $rc[t]$ is the constant coefficient of x^t modulo $x^8 + x^6 + x^5 + x^4 + 1$ which can be obtained via some simple bit x-ors and truncations as the output of a *linear feedback shift register* (LSFR) (see page 16 of FIPS 202).

ι disrupts symmetry.

ι acts only on a few bits in lane $A[0, 0, *]$, but the lane rearrangement π and the slice dispersion ρ ensure that this action affects every lane of A.

Concluding Remarks on SHA-3 and Keccak

Keccak is secure against all known attacks.

In addition to the four hash functions SHA3-m that produce hashes of lengths $m = 224, 256, 384, 512$ using capacities $c = 2m$, the SHA-3 standard supports two other Keccak-based hash functions SHAKE128 and SHAKE256 that produce hashes of the same four lengths m using respective fixed capacities 256 and 512. They are as yet not approved (guidelines for use forthcoming).

To ensure *domain separation*, the SHA3 functions are distinguished from the SHAKE functions by appending different suffixes to the input message M ('01' for SHA3 and '1111' for SHAKE).

Message Authentication Codes (MACs)

A small, fixed-size, key-dependent block that is appended to a message to check data integrity.

- Similar to a hash function, but keyed.

Definition 3 (Message authentication code (MAC))

A single-parameter family $\{C_K\}_{K \in \mathcal{K}}$ of many-to-one functions $C_K : \mathcal{M} \to \{0,1\}^n$ ($n \in \mathbb{N}$) satisfying:

- *Ease of computation:* For any $M \in \mathcal{M}$ and $K \in \mathcal{K}$, $C_K(M)$ is easy to compute.
- *Computation resistance*: for any $K \in \mathcal{K}$, given zero or more message/MAC pairs $(M_i, C_K(M_i))$, it is computationally infeasible to compute any new message/MAC pair $(M, C_K(M))$, $M \neq M_i$ for all i.

Data Integrity using MACs

Computation-resistance implies data integrity (without secrecy):

- Sender and receiver share a secret key K
- Sender computes $MAC = C_K(M)$ and sends (M, MAC) (unencrypted!)
- Receiver computes $MAC' = C_K(M)$ and checks if $MAC' = MAC$. If they match and C_K is computation resistant, the integrity of M is preserved.

Similar to encryption, but (a) no secrecy, (b) MACs need not be reversible, (c) there are many messages with the same MAC.

Sender Authentication using MACs

MACs also provide sender authentication in a similar manner to encryption

- only sender or receiver, who knows K could generate the MAC.

Note: Non-repudiation of data origin not provided

- *either* party possessing K can generate MACs.

Why use MACs (instead of encrypting message plus checksum/hash)?

- Sometimes only integrity is needed (no secrecy).
- Sometimes need integrity to persist longer than the encryption (eg. archival use)

More on MACs

Note 1

MAC should depend equally on all bits of the message. Given valid message/MAC pair, it should still be hard to find another valid pair even if only one bit of the message is modified.

Note 2

Apply first MAC, then encryption to message with MAC appended, rather than vice versa

- $C = E_{K_1}(M \| MAC_{K_2}(M))$ — if encryption is defeated, message integrity is still preserved.
- $(C \| MAC_{K_2}(C))$ with $C = E_{K_1}(M)$ — preserves only integrity of ciphertext which is useless if encryption is defeated

Attacks on MACs

Objectives of adversaries vs. MACs (without prior knowledge of K):

- Compute a new message/MAC pair $(M, C_K(M))$ for some message $M \neq M_i$, given one or more pairs $(M_i, C_K(M_i))$.
- Known-text, chosen-text, and adaptive-chosen-text variations are possible.

Can attack MAC-space or key space:

- Brute-force attack requires effort $\min\{\lceil \frac{m}{n} \rceil 2^m, 2^n\}$ (m-bit MAC, n-bit key)
- As usual, this should be best possible.

CMAC

A secure block cipher (satisfying additional statistical properties) can be used to generate MACs. Two methods are:

1. CBC-MAC:
   - Encrypt the message (zero IV, last block padded with 0s) using CBC mode.
   - The last cipher block (whose bits are dependent on all the key bits and all message bits) is the MAC.
2. CFB-MAC: Same idea as CBC-MAC

A CBC-MAC using DES appears in both FIPS 113 and the ANSI X9.17 standard.

Problem with CBC-MAC

Problem: only secure if messages of *one* fixed length are processed (Bellare, Killian, Rogaway 2000)

Solution (CMAC):

- Use *three* keys, one at each step of the chaining, two for the last block (Black, Rogaway 2000).
- Second two keys may be derived from the encryption key (Iwata, Kurosawa 2003).
- Specified for use with AES and 3DES in NIST Special Pub. 800-38B
- Can be proven secure as long as the underlying block cipher's output is indistinguishable from a random permutation.
- No known weaknesses.

HMAC

Basic idea: $HMAC = H(K_1 \| H(K_2 \| M))$ where H is a cryptographically secure hash function and K is a secret key.

- Bellare, Canetti, Krawczyk (CRYPTO 1996). Complete description in FIPS 198.

Provable security, equivalent to one of:

- computing an output of the compression function of H assuming the IV is unknown,
- finding collisions of the hash function assuming the IV is unknown (birthday attack applies, but more difficult because oracle is required)

Digital Signatures: Definition

Data origin authentication is usually achieved by means of a *signature*, i.e. a means by which the recipient of a message can authenticate the identity of the sender.

Definition 4 (Digital signature)

A means for data authentication that should have two properties:

1. Only the sender can produce his signature.
2. *Anyone* should be easily able to verify the validity of the signature.

Digital Signatures: Observations

Observations:

- Properties 1 and 2 provide *non-repudiation:* if there is a dispute over a signature (a receiver claims that the sender signed the message, whereas the signer claims he didn't), anyone can resolve the dispute. For ordinary written signatures, one might need a hand-writing expert.
- Signatures are different from MACs:
  - both sender and receiver can generate a MAC, whereas only the sender can generate a signature.
  - only sender and receiver can verify a MAC, whereas anyone can verify a signature.
- In order to prevent *replay attacks* (replay a signed message later), it may be necessary to include a time stamp or sequence numbers in the signature.

Signature Capable PKCs

Definition 5 (Signature capability)

A PKC is *signature capable* if $\mathcal{M} = \mathcal{C}$ and $E_{K_1}(D_{K_2}(C)) = C$ for all $C \in \mathcal{C}$.

So in a signature capable PKC, decryptions are right and left inverses (*i.e.* honest-to-goodness inverses) of encryptions.

Example 6

RSA has signature capability. ElGamal and Goldwasser-Micali do not.

Signatures Without Secrecy Using PKC

Alice wishes to send a non-secret message M to Bob along with a signature S that authenticates M to Bob.

She sends (A, M, S) where

- A is her identity,
- M is the message,
- $S = D_A(M)$ is the "decryption" of M under her private key.

To verify S, Bob

- checks A and looks up Alice's public key,
- computes the "encryption" $E_A(S)$ of S under Alice's public key,
- accepts the signature if and only if $M = E_A(S)$

Note that $E_A(S) = E_A(D_A(M)) = M$ if everything was done correctly.

Properties

Anyone can verify a signature since anyone can encrypt under Alice's public key.

In order to forge a signature of a particular message M, Eve would have to be able to do operations using Alice's private key.

Signatures With Secrecy Using PKC

Alice wishes to send an authenticated secret message M to Bob.

She sends $(A, E_B(S, M))$ where A and S are as before and E_B denotes encryption under Bob's public key.

To verify S, Bob decrypts $E_B(S, M)$ and then verifies S as before.

Security of Signatures

Definition 7 (Existential forgery)

A signature scheme is susceptible to *existential forgery* if an adversary can forge a valid signature of another entity for at least one message.

Goals of the attacker:

- total break — recover the private key
- universal forgery — can generate a signature for any message
- selective forgery — can generate a signature for some message of choice
- existential forgery — can generate a signature for at least one message

Existential Forgery on PKC-Generated Signatures

Consider generating a signature S to a message M using a signature-capable PKC as described above.

Eve can create a forged signature from Alice as follows:

1. Selects random $S \in \mathcal{M}$,
2. Computes $M = E_A(S)$,
3. Sends (A, M, S) to Bob.

Bob computes $E_A(S)$ which is M and thus accepts the "signature" S to "message" M.

Usually foiled by language redundancy, but may be a problem is M is random (eg. a cryptographic key).

Preventing Existential Forgery

Solution:

- Alice sends $(A, M, S = D_A(H(M)))$ where H is a public pre-image resistant hash function on $\mathcal{M}$.
- Bob computes $E_A(S)$ and $H(M)$, and accepts the signature if and only if they match.

Foils the attack:

- if Eve generates random S, then she would have to find X such that $H(X) = E_A(S)$ (*i.e.* a pre-image under H), and send (A, X, S) to Bob.
- Bob then computes $E_A(S)$ and compares with $H(X)$.
- Not computationally feasible if H is pre-image resistant.

Existential Forgery if H is not Collision Resistant

Suppose Alice uses a pre-image resistant hash function as described above to sign her messages.

If H is not collision resistant, Eve can forge a signature as follows:

1. Find $M, M' \in \mathcal{M}$ with $M \neq M'$ and $H(M) = H(M')$ (a collision)
2. If S is the signature to M, then S is also the signature to M', as $E_A(S) = H(M) = H(M')$

Note that if Eve intercepts (A, M, S), then she could also find a weak collision M' with $H(M) = H(M')$.

Summary on Signatures via PKC

Use a secure signature capable PKC and a cryptographic (*i.e.* collision resistant) hash function H (security depends on both).

Signing $H(M)$ instead of M also results in faster signature generation if M is long.

H should be a fixed part of the signature protocol, so Eve cannot just substitute H with a cryptographically weak hash function.

GMR-Security

In practice, signature schemes must be resistant to active attacks. We need the equivalent of IND-CCA2 for signatures.

Definition 8 (GMR-security)

A signature scheme is said to be *GMR-secure* if it is existentially unforgeable by a computationally bounded adversary who can mount an adaptive chosen-message attack.

In other words, an adversary who can obtain signatures of any messages of her own choosing from the legitimate signer is unable to produce a valid signature of any new message (for which it has not already requested and obtained a signature) in polynomial time.

GMR stands for *Goldwasser-Micali-Rivest*.

GMR-Secure Versions of RSA

Example 9

RSA-PSS (Probabilistic Signature Scheme), a digital signature analogue of OAEP, is GMR-secure in the random oracle model (ROM) assuming that the RSA problem (computing eth roots modulo n) is hard.

Example 10

RSA with *full-domain hash* — use RSA signatures as usual, signing $H(M)$, but select the hash function H such that $0 \leq H(M) < n$ (n is the RSA modulus) for all messages M.

- Called full-domain because the messages signed are taken from the entire range of possible RSA blocks as opposed to a smaller subrange.
- Also GMR-secure under same assumption as above.

Signature Schemes

Examples of non-PKC-based signature schemes:

- ElGamal — randomized, security based on DLP
- Digital Signature Algorithm — variation of ElGamal with short signatures
- Feige-Fiat-Shamir — security based on computing square roots modulo pq
- Guillou-Quisquater — security based on the RSA problem of computing e-th roots modulo pq

We'll cover the first two here.

Solving General Linear Congruences

We need to solve a general linear congruence of the form

$$ax \equiv b \pmod{m}$$

for $x \in \mathbb{Z}_m^*$, with $m \in \mathbb{N}$ and $a \in \mathbb{Z}_m^*$.

We already saw how to do this for $b = 1$; that's just finding modular inverses.

To solve $ax \equiv b \pmod{m}$ for x : first solve $az \equiv 1 \pmod{n}$ for z using the Extended Euclidean Algorithm. Then $x \equiv bz \pmod{n}$ as

$$ax \equiv a(bz) \equiv (az)b \equiv 1 \cdot b \equiv b \pmod{n} .$$

The El Gamal Signature Scheme

The El Gamal signature scheme is a variation of the El Gamal PKC (same 1985 paper). Security considerations are the same.

A produces her public and private keys as follows:

1. Selects a large prime p and a primitive root g of p.
2. Randomly selects x such that $0 < x < p-1$ and computes $y \equiv g^x \pmod{p}$.

Public key: $\{p, g, y\}$
Private key: $\{x\}$

A also fixes a public cryptographic hash function $H : \{0,1\}^* \mapsto \mathbb{Z}_{p-1}$.

Signing and Verifying

A signs a message $M \in \{0,1\}^*$ as follows:

1. Selects a random integer $k \in \mathbb{Z}^*_{p-1}$.
2. Computes $r \equiv g^k \pmod{p}$, $0 \leq r < p$.
3. Solves $ks \equiv [H(M\|r) - xr] \pmod{p-1}$ for $s \in \mathbb{Z}^*_{p-1}$
4. A's signature is the pair (r, s).

B verifies A's signature (r, s) as follows:

1. Obtains A's authentic public key $\{p, g, y\}$.
2. Verifies that $1 \leq r < p$; if not, reject.
3. Computes $v_1 \equiv y^r r^s \pmod{p}$ and $v_2 \equiv g^{H(M\|r)} \pmod{p}$.
4. Accepts the signature if and only if $v_1 = v_2$.

Proof of Correctness

Proof of correctness.

Note that $ks + rx \equiv H(M, r) \pmod{p-1}$. If the signature (r, s) to message M is valid, then

$$\begin{aligned} v_1 &\equiv y^r r^s \\ &\equiv (g^x)^r (g^k)^s) \\ &\equiv g^{xr+ks} \\ &= g^{H(M\|r)} \\ &\equiv v_2 \pmod{p} \ . \end{aligned}$$

□

Security of ElGamal Signatures

GMR-secure in the ROM assuming that H takes on random values and computing discrete logarithms modulo p is hard.

- Formally, one shows that the DLP reduces to existential forgery, *i.e.* that an algorithm for producing existential forgeries can be used to solve the DLP.

If Step 2 of the verification is omitted (verifying that $r < p$), a universal forgery attack is possible.

- More exactly, if an attacker intercepts a signature (r, s) to a message m, he can forge a signature (R, S) to an *arbitrary* message M.
- The resulting R satisfies $0 \leq R \leq p(p-1)$.

Security of ElGamal Signatures, cont.

The public parameter g must be chosen verifiably at random (eg. publish PRNG, seed, and algorithm used) in order to ensure that g is a primitive root of p

If the same value of k is used to sign two messages, the private key x can be computed with high probability.

The Digital Signature Algorithm (DSA)

Invented by NIST in 1991 and adapted as the *Digital Signature Standard* (DSS) in Dec. 1994.

Variation of El Gamal signature scheme, with similar security characteristics, but much shorter signatures.

DSA Setup

A produces her public and private keys as follows:

1. Selects a 512-bit prime p and a 160-bit prime q such that $q \mid p-1$.
2. Selects a primitive root g of p.
3. Computes $h \equiv g^{(p-1)/q} \pmod{p}$, $0 < h < p$. Note that $h^q \equiv 1 \pmod{p}$ by Fermat's theorem, and if $a \equiv b \pmod{q}$, then $h^a \equiv h^b \pmod{p}$.
4. Randomly selects $x \in \mathbb{Z}$ with $0 < x < q$ and computes $y \equiv h^x \pmod{p}$

Public key: $\{p, q, h, y\}$ ($4 \cdot 512 = 2048$ bits)
Private key: $\{x\}$ (160 bits)

DSA also uses a cryptographically secure hash function $H : \{0,1\}^* \to \mathbb{Z}_q$. The DSS specifies that SHA-1 be used.

Signing and Verifying

A signs message $M \in \{0,1\}^*$ as follows:

1. Selects a random integer k with $0 < k < q$.
2. Computes $r \equiv \left(h^k \pmod{p}\right) \pmod{q}$, $0 < r < q$.
3. Solves $ks \equiv H(M) + xr \pmod{q}$. If $s = 0$, go back to step 1 (this happens with negligible probability).
4. A's signature is the pair $\{r, s\}$ (320 bits, as opposed to 1024)

B verifies A's signature as follows:

1. Obtains A's authentic public key $\{p, q, h, y\}$.
2. Computes the inverse $s^* \in \mathbb{Z}_q^*$ of $s \pmod{q}$.
3. Computes $u_1 \equiv H(M)s^* \pmod{q}$, $u_2 \equiv rs^* \pmod{q}$, and $v \equiv \left(h^{u_1} y^{u_2} \pmod{p}\right) \pmod{q}$, $0 < v < q$.
4. Accepts the signature (r, s) if and only if $v = r$.

Proof of Correctness

Proof of Correctness.

Note that $k \equiv (H(M) + x)s^* \pmod{q}$ and

$$\begin{aligned} v &\equiv h^{u_1} y^{u_2} \\ &\equiv h^{H(M)s^*} y^{rs^*} \\ &\equiv h^{H(M)s^*} h^{xrs^*} \\ &\equiv h^{(H(M)+xr)s^*} \\ &\equiv h^k \equiv r \pmod{p} . \end{aligned}$$

Now v and r are integers strictly between 0 and q that are congruent modulo the much larger modulus p. Hence $v = r$. □

Efficiency of DSA

Small signature (320 bits, much smaller than El Gamal) but the computations are done modulo a 512-bit prime.

Congruence in step 3 of signature generation has a "+" whereas the one in El Gamal has a "−".

The DSA verification procedure is more efficient than the way verification was described for ElGamal

- requires only two modular exponentiations in step 2 as opposed to three in ElGamal.

However, the one in ElGamal can be rewritten in the same efficient way

- check if $ry^{s^*r} \equiv g^{s^*H(M\|r)} \pmod{p}$ where s^* is the inverse of $s \pmod{p-1}$.

Security of DSA

Based on the belief that extracting discrete logs modulo q is hard (seems reasonable).

Proof of GMR-security does *not* hold, because $H(M)$ is signed as opposed to $H(M\|r)$ (reduction to DLP requires that the forger be forced to use the same r for two signatures)

More information: "Another look at provable security" by Koblitz and Menezes, *J. Cryptology* 2007; see "external links" page.

Parameter Sizes for Public-Key Cryptography

Security level: key length for block cipher providing equivalent level of difficulty to break

1024-bit RSA is estimated to provide 80 bits of security

- should be paired with a 160-bit hash function and an 80-bit block cipher (so that all three components equally strong).

Security levels and parameter/key sizes (NIST recommendations):

Security level (in bits)	80	112	128	192	256
Hash size (in bits)	160	224	256	384	512
RSA modulus (in bits)	1024	2048	3072	7680	15360