Lecture Note 7

AUTHENTICATION REQUIREMENTS

Sourav Mukhopadhyay

Cryptography and Network Security - MA61027

- In the context of communications across a network, the following attacks can be identified:
  1. **Disclosure:** Release of message contents to any person or process not possessing the appropriate cryptographic key.
  2. **Traffic analysis:** Discovery of the pattern of traffic between parties. In a connection-oriented application, the frequency and duration of connections could be determined. In either a connection-oriented or connectionless environment, the number and length of messages between parties could be determined.

3. **Masquerade:** Insertion of messages into the network from a fraudulent source. This includes the creation of messages by an opponent that are purported to come from an authorized entity. Also included are fraudulent acknowledgments of message receipt or nonreceipt by someone other than the message recipient.
4. **Content Modification:** Changes to the contents of a message, including insertion, deletion, transposition, or modification.
5. **Sequence modification:** Any modification to a

sequence of messages between parties, including insertion, deletion, and reordering.

6. **Timing modification:** Delay or replay of messages. In a connection-orientated application, an entire session or sequence of messages could be a replay of some previous valid session, or individual messages in the sequence could be delayed or replayed.

7. **Repudiation:** Denial of receipt of message by destination or denial of transmission of message by source.

- **Message authentication** is a procedure to verify that received messages come from the alleged source and have not been altered.
- Message authentication may also verify sequencing and timeliness.
- A **digital signature** is an authentication technique that also includes measures to counter repudiation by either source or destination.
- Any message authentication or digital signature mechanism can be viewed as having fundamentally two levels.

- At the lower level, there must be some sort of function that produces an authenticator: a value to be used to authenticate a message.

- This lower-level function is then used as primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a message.

- This section is concerned with the types of functions that may be used to produce an authenticator. These functions may be grouped into three classes, as follows:

  1. **Message Encryption:** The ciphertext of the entire message serves as its authenticator.

2. **Message Authentication Code (MAC):** A public function of the message and a secret key that produces a fixed length value that serves as the authenticator.
3. **Hash Functions:** A public function that maps a message of any length into a fixed length hash value, which serves as the authenticator.

- We will mainly be concerned with the last class of function however it must be noted that hash functions and MACs are very similar except that a hash code doesn't require a secret key.

- With regard to the first class, this can be seen to provide authentication by virtue of the fact that only the sender and receiver know the key.
- Therefore the message could only have come from the sender.
- However there is also the problem that the plaintext message should be recognisable as plaintext message (for example if it was some sort of digitised X-rays it mightn't be).

Hash Functions

- A hash value is generated by a function H of the form:

$$h = H(M)$$

- Where M is a variable-length message, and $H(M)$ is the fixed length hash value (also referred to as a message digest or hash code).

- Figures 1 and 2 shows the basic uses of the hash function whereas figure 3 shows the general structure of a hash code.

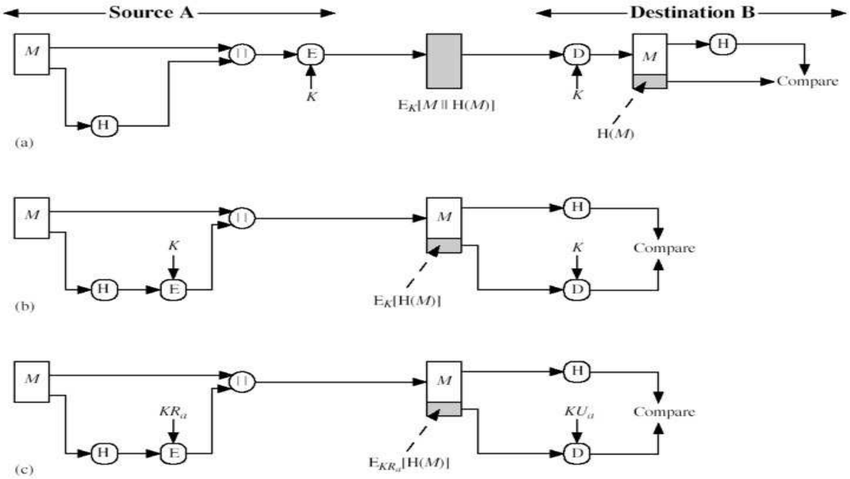


Figure 1: Basic uses of the hash function.

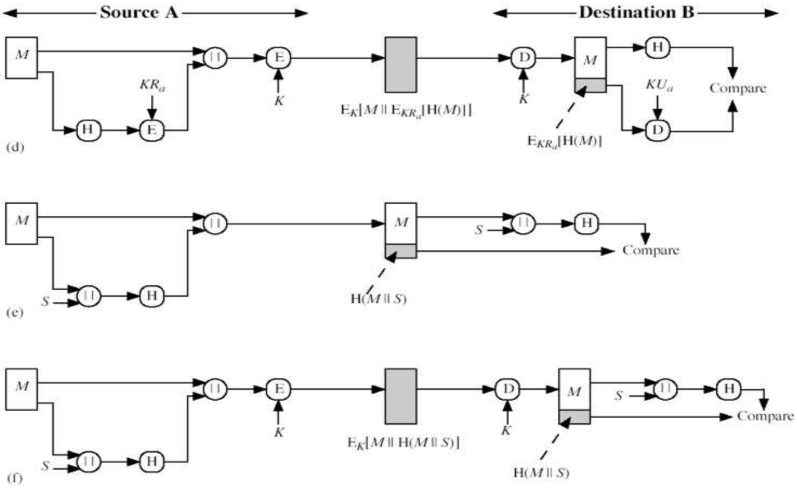


Figure 2: Basic uses of the hash function (cont.).

- The hash value is appended to the message at the source at the time when the message is assumed or known to be correct.
- The receiver authenticates that message by recomputing the hash value.
- Because the hash function itself is not considered to be secret, some means is required to protect the hash value (see figures 1 and 2).
- We begin by examining the requirements for a hash function to be used for message authentication.

- The purpose of a hash function is to produce a "fingerprint" of a file, message, or other block of data.
- To be useful for message authentication, a hash function H must have the following properties:

  1. H can be applied to a block of data of any size.
  2. H produces a fixed-length output.
  3. $H(x)$ is relatively easy to compute for any given x, making both hardware and software implementations practical.
  4. For any given code h, it is computationally infeasible

to find x such that $H(x) = h$.

5. For any given block x, it is computationally infeasible to find $y \neq x$ with H(y) = H(x) (sometimes referred to as **weak collision property**).
6. It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$ (sometimes referred to as **strong collision property**).

- The first three properties are requirements for the practical application of a hash function to message authentication.
- The fourth property is the “one-way” property; it is easy to generate a code given a message but virtually impossible to generate a message given a code.
- This property is important if the authentication technique involves the use of a secret value (see figure 2e).
- The secret value itself is not sent; however, if the hash function is not one-way, an attacker can easily discover the secret value.

- If the attacker can observe or intercept a transmission, the attacker obtains the message M and the hash code $C = H(S_{AB}||M)$.

- The attacker then inverts the hash function to obtain $S_{AB}||M = H^{-1}(C)$.

- Because the attacker now has both M and $S_{AB}||M$, it is a trivial matter to recover S_{AB}.

- The fifth property guarantees that an alternative message hashing to the same value as a given message cannot be found.

- This prevents forgery when an encrypted hash code is used (see figures 1b and c).
- For these cases, the opponent can read the message and therefore generate its hash code. But, because the opponent does not have the secret key, the opponent should not be able to alter the message without detection.
- If this property were not true, an attacker would be capable of the following sequence:
    1. Observe or intercept a message plus its encrypted hash code.

2. Generate an unencrypted hash code from the message.
3. Generate an alternate message with the same hash code.

- A hash function that satisfies the first five properties in the preceding list is referred to as a **weak hash function**.

- If the sixth property is also satisfied, then it is referred to as a **strong hash function**.

- The sixth property protects against a sophisticated class of attack known as the **birthday attack** which we will be looking at later in the notes.

- Figure 3 shows the general structure of a secure hash code.

- In the next section we are going to study a specific algorithm (SHA-1) which will be seen to have this format.
- Notice this has a similar structure to the CBC mode used for symmetric algorithms.

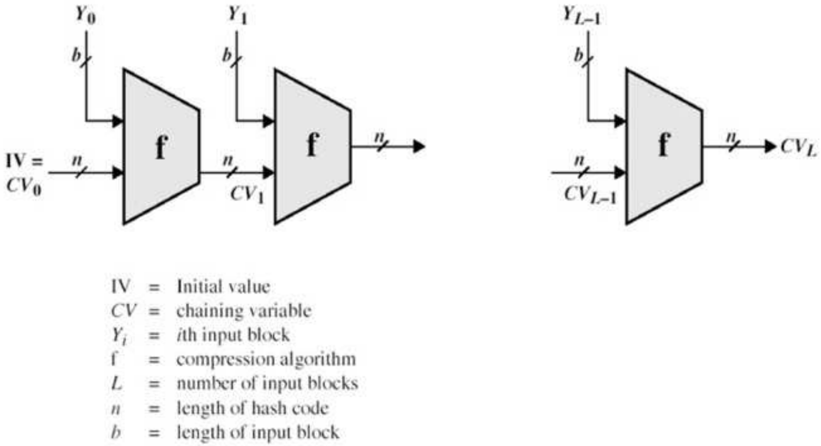

Y_0
b
Y_1
b
Y_{L-1}
b
f
f
f
IV =
CV_0
n
n
CV_1
n
n
CV_{L-1}
n
CV_L
IV = Initial value
CV = chaining variable
Y_i = ith input block
f = compression algorithm
L = number of input blocks
n = length of hash code
b = length of input block

Figure 3: General Structure of Secure Hash Code.

The Secure Hash Algorithm

- The **Secure Hash Algorithm (SHA)** was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993; a revised version was issued as FIPS 180-1 in 1995 and is generally referred to as **SHA-1**.
- The actual standards document is entitled Secure Hash Standard.

- SHA is based on the MD4 algorithm which is a message digest algorithm that was developed by Ron Rivest at MIT (the “R” in the RSA (Rivest-Shamir-Adelman) public key encryption algorithm).

- MD4 was later replaced with the popular MD5 algorithm also by Ron Rivest however advances in cryptanalysis and computing power have led to their decline in popularity.
- Both MD4 and MD5 produce a 128 bit message digest whereas SHA-1 produces a 160 bit as will be seen.
- In 2002, NIST produced a revised version of the standard, FIPS180-2, that defined three new versions of SHA.
- These have hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512 respectively.

- These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1.
- In 2005, NIST announced the intention to phase out approval of SHA-1 and move to a reliance on the other SHA versions by 2010.
- Shortly thereafter, a research team described an attack in which two separate messages could be found that deliver the same SHA-1 hash using 2^{69} operations.

- SHA-1 takes as input a message with a maximum length of less than 2^{64} bits and produces as output a 160 bit message digest.
- The input is processed in 512-bit blocks.
- Figure 4 depicts the overall processing of a message to produce a digest.
- Although this diagram has MD5 as the hash function the structure is exactly the same for SHA-1 with the exception that the message length is limited in size (its isn't for MD5) and the hash value (and intermediate values CV_i) are 160 bits and not 128 as shown (which is the case for MD5).

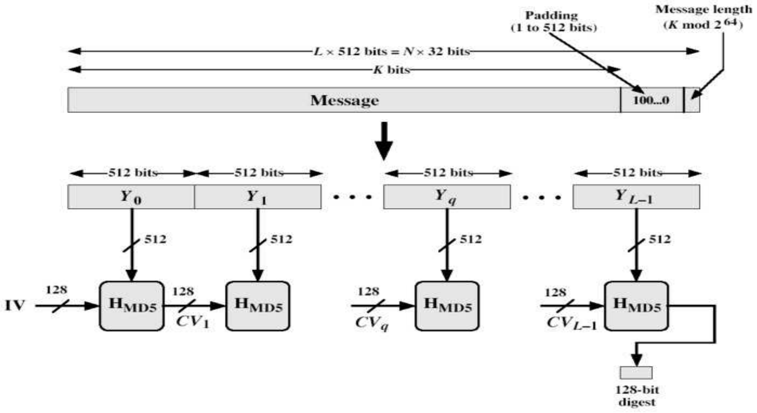


Figure 4: Message Digest Generation Using MD5 (equally applicable to SHA-1 with 160 bits instead of 128 etc.).

5 steps: 1). Append padding bits

- The message is padded so that its length is congruent to 448 modulo 512 (length $\equiv 448 \pmod{512}$).
- That is, the length of the padded message is 64 bits less than an integer multiple of 512 bits.
- Padding is always added, even if the message is already of the desired length.
- Thus, the number of padding bits is in the range of 1 to 512 bits.
- The padding consists of a single 1-bit followed by the necessary number of 0-bits.

2). Append length

- A block of 64 bits is appended to the message.
- This block is treated as an unsigned 64-bit integer (most significant byte first) and contains the length of the original message (before padding).

3). Initialize MD buffer

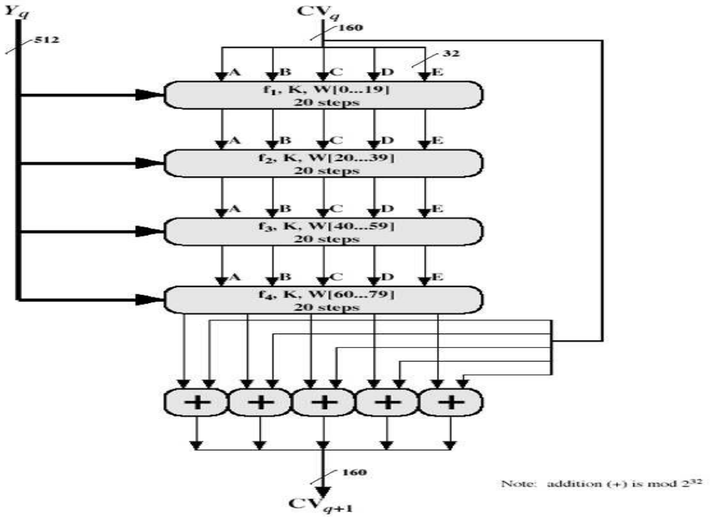
- A 160 bit buffer is used to hold intermediate values and final results of the Hash function represented as 5, 32 bit registers (A, B, C, D, E) initialized as follows:

$$\begin{aligned} A &= 67452301 \\ B &= EFCDAB89 \\ C &= 98BADCFE \\ D &= 10325476 \\ E &= C3D2E1F0 \end{aligned}$$

4). Process message in 512 bit (16 word) blocks

- The heart of the algorithm is a module which consists of four "rounds" of processing of 20 steps each (see figure 5).
- Each round has similar structure but uses a different primitive logical function (f_1, f_2, f_3 and f_4).
- Each round takes as input the current 512-bit block being processed (Y_q) and the 160-bit buffer value ABCDE and updates the contents of the buffer.

- Each round also makes use of an additive constant K_t where $0 \leq t \leq 79$ indicates one of the 80 steps across four rounds.
- In fact, only four distinct constants are used (one for $0 \leq t \leq 19$, $20 \leq t \leq 39$, $40 \leq t \leq 59$ and $60 \leq t \leq 79$).
- The output of the fourth round is added (modulo 2^{32}) to the input to the first round (CV_q) to produce CV_{q+1}.



Note: addition (+) is mod 2^{32}

Figure 5: SHA-1 Processing of a Single 512-bit Block (SHA-1 compression function).

5). Output

- After all L 512 bit blocks have been processed the output from the Lth stage is the 160 bit digest.

- We can summarise the behavior SHA-1 as follows:

$$\begin{aligned}
\mathrm{CV}_0 &= \mathrm{IV} \\
\mathrm{CV}_{q+1} &= \mathrm{SUM}_{32}(CV_q, \mathrm{ABCDE}_q) \\
\mathrm{MD} &= \mathrm{CV}_L
\end{aligned}$$

IV = initial value of the ABCDE buffer, defined in step 3.

ABCDE_q = the output of the last round of processing of the qth message block.

L = the number of blocks in the message (including padding and length fields).

SUM_{32} = Addition modulo 2^{32} performed separately on each word of the pair of inputs.

MD = final message digest value.

The next frontier, SHA-3

- SHA-3 Next generation hash algorithms
- Call for a new crop of hash
  - November 2007
  - http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf
  - Deadline October 2008
  - Roughly based on AES competition

Digital Signatures

- Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other.
- Several forms of dispute between the two are possible. For example, suppose that John sends an authenticated message to Mary using one of the schemes described earlier.
- Consider the following disputes that could arise:
  - Mary may forge a different message and claim that it came from John. Mary would simply have to create a message and append an authentication code using the key that John and Mary share.

- John can deny sending the message. Because it is possible for Mary to forge a message, there is no way to prove that John did in fact send the message.

- Both scenarios are of legitimate concern. In situations where there is not complete trust between sender and receiver, something more than authentication is needed.
- The most attractive solution to this problem is the **digital signature**.

- The digital signature is analogous to the handwritten signature.
- It must have the following properties:
  - It must verify the author and the date and time of the signature.
  - It must authenticate the contents at the time of the signature.
  - It must be verifiable by third parties, to resolve disputes.

- Thus, the digital signature function includes the authentication function. On the basis of these properties, we can formulate the following requirements for a digital signature:
    - The signature must be a bit pattern that depends on the message being signed.
    - The signature must use some information unique to the sender, to prevent both forgery and denial.
    - It must be relatively easy to produce the digital signature.

- It must be relatively easy to recognise and verify the digital signature.
- It must be computationally infeasible to forge a digital signature, either by constructing a new message for an existing digital signature or by constructing a fraudulent digital signature for a given message.
- It must be practical to retain a copy of the digital signature in storage.

Signature Scheme

- A signature scheme is a five-tuple $(\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}, \mathcal{V})$, where the following conditions are satisfied:

  1. $\mathcal{P}$ is a finite set of all possible messages.
  2. $\mathcal{A}$ is a finite set of all possible signatures.
  3. $\mathcal{K}$, the keyspace, is a finite set of possible keys.
  4. $\mathcal{S}$ is a set of all possible signature algorithms.
  5. $\mathcal{V}$ is a set of all possible verification algorithms.

- For each $K \in \mathcal{K}$, there is a signing algorithm $sig_k \in \mathcal{S}$ and a corresponding verification algorithm $ver_k \in \mathcal{V}$. Each $sig_K : \mathcal{P} \rightarrow \mathcal{A}$ and $ver_k : \mathcal{P} \times \mathcal{A} \rightarrow \{\text{true, false}\}$ are functions such that the following equation is satisfied for every message $x \in \mathcal{P}$ and for every signature $y \in \mathcal{A}$:

$$\begin{aligned} ver(x, y) &= \text{true if } \ y = sig(x) \\ &= \text{false if } \ y \neq sig(x) \end{aligned}$$

- For every $K \in \mathcal{K}$, the function sig_k and ver_k should be polynomial-time functions. ver_k will be a public function and sig_k will be secret.

RSA Signature Scheme

- Let $n = pq$, where p and q are primes. Let $\mathcal{P} = \mathcal{A} = Z_n$, and define

$$\mathcal{K} = \{(n, p, q, a, b) : n = pq \text{ and } ab \equiv 1 \pmod{\phi(n)}\}$$

- The values n and b are public, and the value p, q, a are secret.

- $K = (n, p, q, a, b)$, define

$$sig_K(x) = x^a \bmod n$$

and

$$ver_K(x, y) = true \iff x \equiv y^b \pmod{n}$$

$(x, y \in Z_n)$.

ElGamal Signature Scheme

- Let p be a prime such that the discrete log problem in Z_p is intractable, and let $\alpha \in Z_p^*$ be a primitive element.
- Let $\mathcal{P} = Z_p^*$, $\mathcal{A} = Z_p^* \times Z_{p-1}$, and define

$$\mathcal{K} = \{(p, \alpha, a, \beta) : \beta \equiv \alpha^a \pmod{p}\}.$$

- The values p, α and β are public, and a is secret.

- $K = (p, \alpha, a, \beta)$, for a (secret) random number $k \in Z_{p-1}^*$, define

$$sig_K(x, k) = (\gamma, \delta),$$

where

$$\gamma = \alpha^k \bmod p$$

and

$$\delta = (x - a\gamma)k^{-1} \bmod (p-1).$$

- For $x, \gamma \in Z_p^*$ and $\delta \in Z_{p-1}$, define

$$ver_K(x, \gamma, \delta) = true \iff \beta^\gamma \gamma^\delta \equiv \alpha^x \pmod{p}.$$

Example

- Suppose $p = 467, \alpha = 2, a = 127$, and hence

$$\beta = \alpha^a \bmod p = 2^{127} \bmod 467 = 132.$$

- Suppose Bob wants to sign the message $x = 100$ and he chooses the random value $k = 213$ (note that $gcd(213, 466) = 1$ and $213^{-1} \bmod 466 = 431$). Then

$$\gamma = 2^{213} \bmod 467 = 29$$

and

$$\delta = (100 - 127 \times 29)431 \bmod 466 = 51.$$

- Anyone can verify this signature by checking that

$$132^{29}29^{51} \equiv 189 \pmod{467}$$

and

$$2^{100} \equiv 189 \pmod{467}.$$

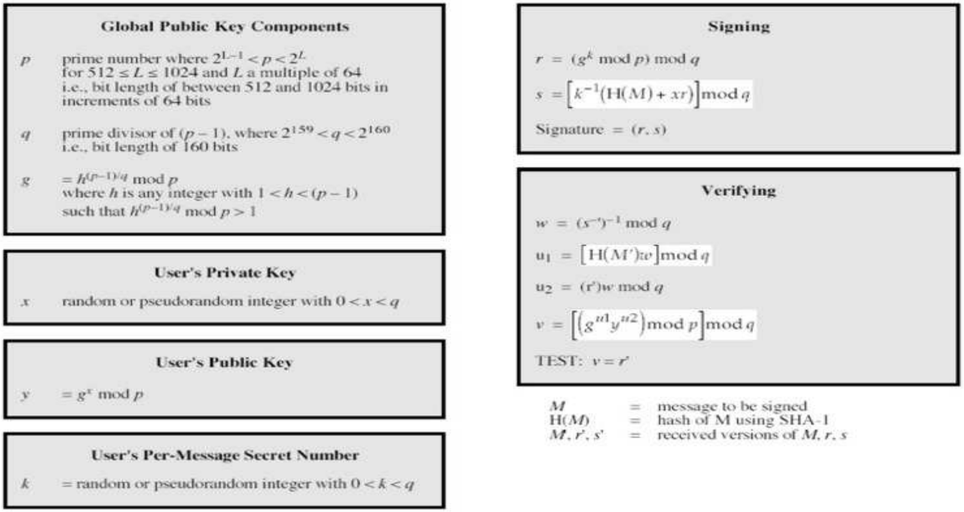
- Hence, the signature is valid.

Digital Signature Standard (DSS)

- The National Institute of Standards and Technology (NIST) has published FIPS 186 known as the Digital Signature Standard (DSS).
- The DSS makes use of the Secure Hash Algorithm (SHA) that we just discussed and presents a new digital signature technique, the Digital Signature Algorithm (DSA).
- The DSS was originally proposed in 1991 and revised in 1993 in response to public feedback concerning the security of the scheme.

- There was a further minor revision in 1996. In 2000, an expanded version of the standard was issued as FIPS 186-2. This latest version also incorporates digital signature algorithms based on RSA and on elliptic curve cryptography.
- Here, we discuss the original DSS algorithm.
- The DSS uses an algorithm that is designed to provide only the digital signature function.
- Unlike RSA, it cannot be used for encryption or key exchange.

- Nevertheless, it is a public-key technique. It is based on the difficulty of computing discrete logarithms (as is the Diffie Hellman key exchange).



Global Public Key Components

p prime number where $2^{L-1} < p < 2^L$
for $512 \le L \le 1024$ and L a multiple of 64
i.e., bit length of between 512 and 1024 bits in increments of 64 bits

q prime divisor of $(p-1)$, where $2^{159} < q < 2^{160}$
i.e., bit length of 160 bits

g $= h^{(p-1)/q} \bmod p$
where h is any integer with $1 < h < (p-1)$
such that $h^{(p-1)/q} \bmod p > 1$

User's Private Key

x random or pseudorandom integer with $0 < x < q$

User's Public Key

y $= g^x \bmod p$

User's Per-Message Secret Number

k = random or pseudorandom integer with $0 < k < q$

Signing

$r = (g^k \bmod p) \bmod q$

$s = \left[k^{-1}(\mathrm{H}(M) + xr)\right] \bmod q$

Signature = (r, s)

Verifying

$w = (s')^{-1} \bmod q$

$u_1 = \left[\mathrm{H}(M')w\right] \bmod q$

$u_2 = (r')w \bmod q$

$v = \left[\left(g^{u1}y^{u2}\right) \bmod p\right] \bmod q$

TEST: $v = r'$

M	=	message to be signed
$\mathrm{H}(M)$	=	hash of M using SHA-1
M', r', s'	=	received versions of M, r, s

Figure 6: The Digital Signature Algorithm (DSA).

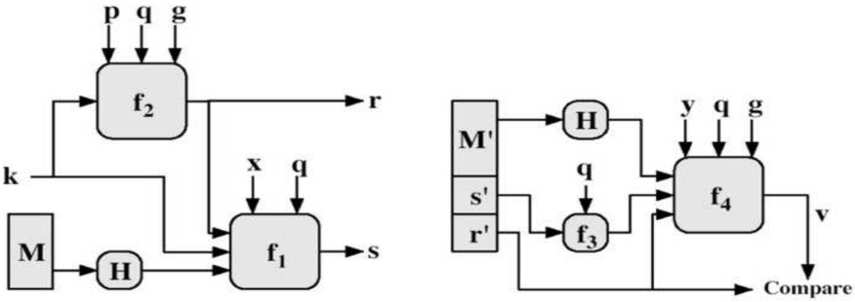
- The overall scheme is seen in figure 6.
- There are three parameters that are public and can be common to a group of users.
  1. A 160-bit prime number q is chosen.
  2. Next, a prime number p is selected with a length between 512 and 1024 bits such that q divides $(p-1)$.
  3. Finally, g is chosen to be of the form $h^{(p-1)/q}$ mod p, where h is an integer between 14 and $(p-1)$ with the restriction that g must be greater than 1.
- With these numbers in hand, each user selects a private key and generates a public key.

- The private key x must be a number from 1 to $(p-1)$ and should be chosen randomly or pseudorandomly.
- The public key is calculated from the private key as $y = g^x \bmod p$. The calculation of y is relatively straightforward. However finding x given the other parameters appears not to be.

- To create a signature, a user calculates two quantities, r and s, that are functions of the public key components (p, q, g), the user's private key (x), the hash code of the message H(M), and an additional integer k that should be generated randomly or pseudorandomly and be unique for each signing.
- At the receiving end, verification is performed using the formulas shown in Figure 6.
- The receiver generates a quantity v that is a function of the public key component, the sender's public key, and the hash code of the incoming message.

- If this quantity matches the r component of the signature, then the signature is validated.
- Figure 7 depicts the functions of signing and verifying.
- Note that the test at the end is on the value r, which does not depend on the message at all.
- Instead, r is a function of k and the three global public-key components.
- The multiplicative inverse of k mod q is passed to a function that also has as inputs the message hash code and the user's private key.

- The structure of this function is such that the receiver can recover r using the incoming message and signature, the public key of the user, and the global public key.
- Given the difficulty of taking discrete logarithms, it is infeasible for an opponent to recover K from r or to recover x from s.



$s = f_1(H(M), k, x, r, q) = (k^{-1}\,(H(M) + xr)) \bmod q$

$r = f_2(k, p, q, g) = (g^k \bmod p) \bmod q$

(a) Signing

$w = f_3(s', q) = (s')^{-1} \bmod q$

$v = f_4(y, q, g, H(M'), w, r')$

$= ((g^{(H(M')w) \bmod q}\, y^{r'w \bmod q}) \bmod p) \bmod q$

(b) Verifying

Figure 7: DSS Signing and Verifying.