LECTURE 6

HMAC and EDS

Telecommunication systems department

Lecturer: assistant professor Persikov Anatoliy Valentinovich

NEED FOR INTEGRITY

Verifying the integrity and authenticity of information is a prime necessity in computer systems and networks. In particular, two parties communicating over an insecure channel require a method by which information sent by one party can be validated as authentic (or unmodifed) by the other.

Most commonly such a mechanism is based on a secret key shared between the parties and takes the form of a **Message Authentication Code** (MAC). (Other terms used include "**Integrity Check Value**" or "**cryptographic checksum**").

In this case, when party A transmits a message to party B, it appends to the message a value called the **authentication tag**, computed by the MAC algorithm as a function of the transmitted information and the shared secret key. At reception, B recomputes the authentication tag on the received message using the same mechanism (and key) and checks that the value he obtains equals the tag attached to the received message. Only if the values match is the information received considered as not altered on the way from A to B. The goal is to prevent forgery, namely, the computation, by the adversary, of a message (not sent by the legitimate parties) and its corresponding valid authentication tag.

THE NESTED CONSTRUCTION NMAC

Let $k = (k_1, k_2)$ where k_1 and k_2 are keys to the function F (i.e., random strings of length l each). We define a MAC function $NMAC(x)$ which works on inputs x of arbitrary length as

$$NMAC_k(x) = F_{k_1}(F_{k_2}(x)).$$

Notice that the outer function acts on the output of the iterated function and then it involves only one iteration of the compression function. That is, this outer function is basically the compression function f_{k_1} acting on $F_{k_2}(x)$ padded to a full block size (in some standard way as defined by the underlying hash scheme F).

Notice the **simplicity** and **efficiency** of the construction. The cost of the internal function is exactly the same as hashing the data with the basic (keyless hash function).

The only **additional cost** is the outer application which, as said, involves only one iteration of the compression function.

THE FUNCTION HMAC

Let F be the (iterated and key-less) hash function initialized with its usual fixed IV. The function $HMAC$ works on inputs x of arbitrary length and uses a single random string k of length l as its key:

$$HMAC_k(x) = F(\bar{k} \oplus opad;\ F(\bar{k} \oplus ipad;\ x))$$

where

$\bar{k}$ is the completion by adding 0's of k to a full b-bit block-size of the iterated hash function,

$opad$ and $ipad$ are two fixed b-bits constants (the "i" and "o" are mnemonics for inner and outer),

$\oplus$ is the bitwise Exclusive Or operator, and the commas represent concatenation of the information.

$opad$ is formed by repeating the byte x'36' as many times as needed to get a b-bit block, and $ipad$ is defined similarly using the byte x'5c'. (For example, in the case of MD5 and SHA-1 these bytes are repeated 64 times).

SECURITY OF HMAC

The security of HMAC is based on the security of NMAC.

The main observation for relating these two functions and their security is that by defining $k_1 = f(\bar{k} \oplus opad)$ and $k_2 = f(\bar{k} \oplus ipad)$, we get that $HMAC_k(x) = NMAC_{(k_1,k_2)}(x)$.

In other words, the above transformation on the key makes HMAC a particular case of NMAC, where the keys k_1 and k_2 are "pseudorandomly" derived from k using the compression function f. Since the analysis of NMAC assumes that k_1 and k_2 are random and independently chosen keys, then in order to apply this analysis to HMAC one needs to assume that k_1 and k_2 derived using f cannot be distinguished by the attacker from truly random keys. This represents an additional assumption on the quality of the function f (keyed through the input k) as a pseudorandom function. We require a relatively weak form of pseudorandomness since the adversary trying to learn about possible dependencies of k_1 and k_2 does not get to see directly the output of the pseudorandom function on any input.

To sum things up, attacks that work on HMAC and not on NMAC are possible, in principle. However, such an attack would reveal major weaknesses of the pseudorandom properties of the underlying hash function.

SECURITY OF HMAC

It is important to note that in practice most keys are chosen pseudorandomly rather than as truly random strings; in particular, it is plausible that even if one uses NMAC, implementations will choose to derive k_1 and k_2 using a pseudorandom generator. In the case of HMAC such a pseudorandom generator is "built-in" through the definition of the function using the function f and the above defined pads. This use for pseudorandom generation of functions like MD5 or SHA-1 is very common in practical implementations.

The above particular values of $opad$ and $ipad$ were chosen to have a very simple representation (to simplify the function's specification and minimize the potential of implementation errors), and to provide a high Hamming distance between the pads. The latter is intended to exploit the mixing properties attributed to the compression function underlying the hash schemes in use. These properties are important in order to provide computational independence between the two derived keys.

Finally, we note that the use of a single l-bit long key as opposed to two (independent) keys does not represent a weakening of the function relative to exhaustive search of the key, since even when chosen independently the keys k_1 and k_2 can be individually searched through a divide and conquer attack.

BIRTHDAY ATTACKS

Birthday attacks, that are the basis to finding collisions in cryptographic hash functions, can be applied to attack also keyed MAC schemes based on iterated functions (including also CBC-MAC, and other schemes).

These attacks apply to our new constructions as well. In particular, they constitute the best known forgery attacks against both the NMAC and HMAC constructions.

Consideration of these attacks is important since they strongly improve on naive exhaustive search attacks. However, their practical relevance against these functions is negligible given the typical hash lengths like 128 or 160, since these attacks require knowledge of the MAC value (for a given key) on about $2^{l/2}$ messages (where l is the length of the hash output). For values of $l \geq 128$ the attack becomes totally infeasible. In contrast to the birthday attack on key-less hash functions, the new attacks require interaction with the key owner to produce the MAC values on a huge number of messages, and then allow for no parallelization. For example, when using MD5 such an attack would require the authentication of 2^{64} blocks (or 2^{73} bits) of data using the same key. On a 1 Gbit/sec communication link, one would need 250,000 years to process all the data required by such an attack. This is in sharp contrast to birthday attacks on key-less hash functions which allow for far more efficient and close-to-realistic attacks.

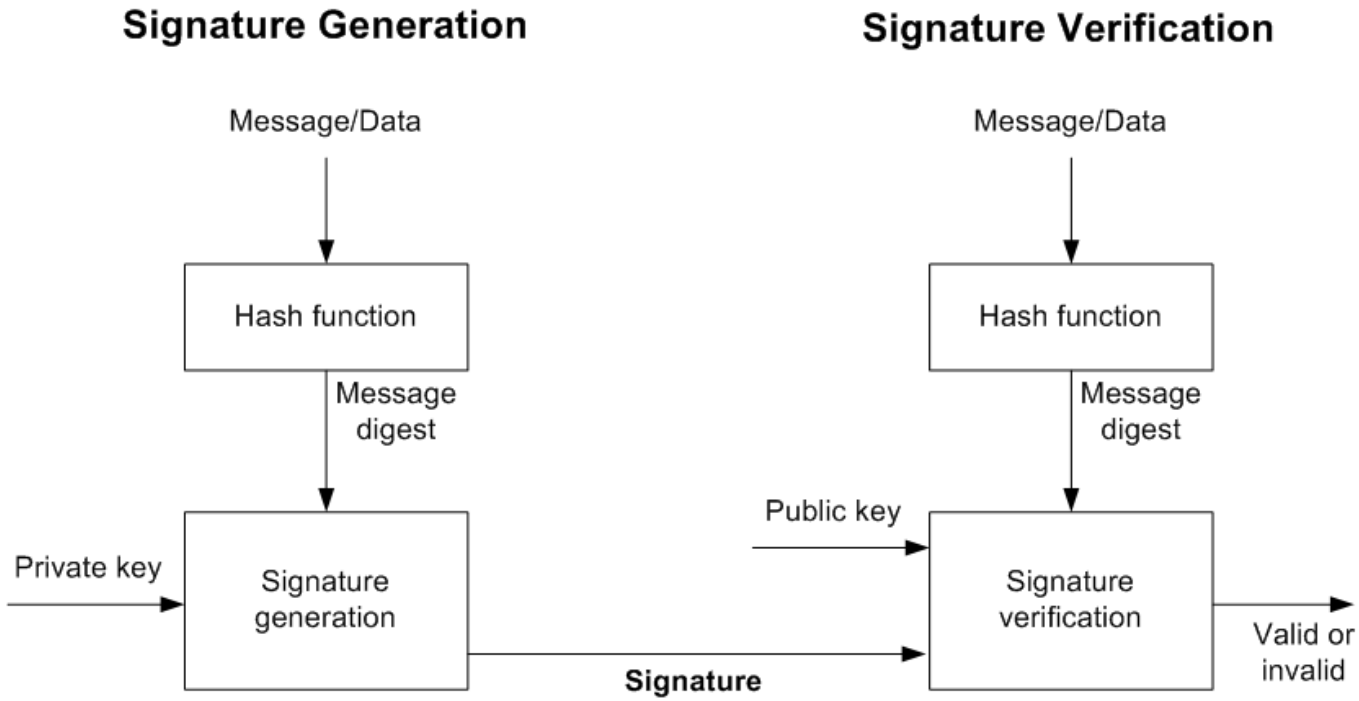
BIRTHDAY ATTACKS

Notice that these attacks produce forgery of the MAC function but not key recovery. In some versions of the envelope method (the case where the same key is used to prepend and append and no block alignment of the appended key is performed), the birthday attacks can be further enhanced to provide full key recovery in time much shorter than required by full exhaustive search. Since these attacks require at least the complexity mentioned above for forgery based on birthday attacks, they cannot be considered as practical ones. Yet, it is interesting to note that they do not apply to either of our constructions, since here the alignment issue exploited by these attacks is no applicable.

The forms of birthday attacks that apply to our constructions can become feasible only if very significant weaknesses in the collision probability of the underlying hash function are discovered. However, in such a case the basic use of such a function as collision-resistant (as originally intended) would be strongly compromised, and the function should be dropped for cryptographic use. Finally, we mention that these birthday attacks (at least in their straightforward form) can be avoided by randomizing the MAC construction in a per-message basis.

DIGITAL SIGNATURE

A **digital signature** is an electronic analogue of a written signature; the digital signature can be used to provide assurance that the claimed signatory signed the information.



In addition, a digital signature may be used **to detect whether or not the information was modified after it was signed** (i.e., to detect the integrity of the signed data). These assurances may be obtained whether the data was received in a transmission or retrieved from storage.

DIGITAL SIGNATURE

A **digital signature algorithm** includes a **signature generation process** and a **signature verification process**.

A **signatory** uses the generation process to generate a digital signature on data; a verifier uses the verification process to verify the authenticity of the signature. Each signatory has a public and private key and is the owner of that key pair. The private key is used in the signature generation process. The key pair owner is the only entity that is authorized to use the private key to generate digital signatures. In order to prevent other entities from claiming to be the key pair owner and using the private key to generate fraudulent signatures, the private key must remain secret. The approved digital signature algorithms are designed to prevent an adversary who does not know the signatory's private key from generating the same signature as the signatory on a different message. In other words, signatures are designed so that they cannot be forged. A number of alternative terms are used in this Standard to refer to the signatory or key pair owner. An entity that intends to generate digital signatures in the future may be referred to as the intended signatory. Prior to the verification of a signed message, the signatory is referred to as the claimed signatory until such time as adequate assurance can be obtained of the actual identity of the signatory.

DIGITAL SIGNATURE

The **public key** is used in the signature verification process. The public key need not be kept secret, but its integrity must be maintained. Anyone can verify a correctly signed message using the public key.

For both the signature generation and verification processes, the **message** (i.e., the signed data) **is converted to a fixed-length representation of the message** by means of an approved hash function. Both the original message and the digital signature are made available to a verifier.

A **verifier requires** assurance that the public key to be used to verify a signature belongs to the entity that claims to have generated a digital signature (i.e., the claimed signatory). That is, a verifier requires assurance that the signatory is the actual owner of the public/private key pair used to generate and verify a digital signature. A binding of an owner's identity and the owner's public key shall be effected in order to provide this assurance.

A **verifier** also **requires** assurance that the key pair owner actually possesses the private key associated with the public key, and that the public key is a mathematically correct key.

By obtaining these assurances, the verifier has assurance that if the digital signature can be correctly verified using the public key, the digital signature is valid (i.e., the key pair owner really signed the message). Digital signature validation includes both the (mathematical) verification of the digital signature and obtaining the appropriate assurances.

DIGITAL SIGNATURE

The following are reasons why such assurances are required.

1. If a verifier does not obtain assurance that a signatory is the actual owner of the key pair whose public component is used to verify a signature, the problem of forging a signature is reduced to the problem of falsely claiming an identity.
2. If the public key used to verify a signature is not mathematically valid, the arguments used to establish the cryptographic strength of the signature algorithm may not apply. The owner may not be the only party who can generate signatures that can be verified with that public key.
3. If a public key infrastructure cannot provide assurance to a verifier that the owner of a key pair has demonstrated knowledge of a private key that corresponds to the owner's public key, then it may be possible for an unscrupulous entity to have their identity (or an assumed identity) bound to a public key that is (or has been) used by another party. The unscrupulous entity may then claim to be the source of certain messages signed by that other party. Or, it may be possible that an unscrupulous entity has managed to obtain ownership of a public key that was chosen with the sole purpose of allowing for the verification of a signature on a specific message.

THE DIGITAL SIGNATURE ALGORITHM (DSA)

A DSA digital signature is computed using a set of domain parameters, a private key x, a per-message secret number k, data to be signed, and a hash function. A digital signature is verified using the same domain parameters, a public key y that is mathematically associated with the private key x used to generate the digital signature, data to be verified, and the same hash function that was used during signature generation.

These parameters are defined as follows:

p a prime modulus, where $2^{L-1} < p < 2^L$, and L is the bit length of p.

q a prime divisor of $(p - 1)$, where $2^{N-1} < q < 2^N$, and N is the bit length of q.

g a generator of the subgroup of order $q \bmod p$, such that $1 < g < p$.

x the private key that must remain secret; x is a randomly or pseudorandomly generated integer, such that $0 < x < q$, i.e., x is in the range $[1, q-1]$.

y the public key, where $y = g^x \bmod p$.

k a secret number that is unique to each message; k is a randomly or pseudorandomly generated integer, such that $0 < k < q$, i.e., k is in the range $[1, q-1]$.

THE DIGITAL SIGNATURE ALGORITHM (DSA)

Selection of parameter sizes and hash functions for DSA

Standard FIPS 186-3 specifies the following choices for the pair L and N (the bit lengths of p and q, respectively):

$L = 1024, N = 160$

$L = 2048, N = 224$

$L = 2048, N = 256$

$L = 3072, N = 256$

DSA per-message secret number

A new secret random number k shall be generated prior to the generation of each digital signature for use during the signature generation process. This secret number shall be protected from unauthorized disclosure and modification.

k^{-1} is the multiplicative inverse of k with respect to multiplication modulo q; i.e., $0 < k^{-1} < q$ and $1 = (k^{-1} \cdot k) \bmod q$. This inverse is required for the signature generation process. k and k^{-1} may be pre-computed, since knowledge of the message to be signed is not required for the computations. When k and k^{-1} are pre-computed, their confidentiality and integrity shall be protected.

THE DIGITAL SIGNATURE ALGORITHM (DSA)

DSA signature generation

Let N be the bit length of q. Let $\min(N, outlen)$ denote the minimum of the positive integers N and $outlen$, where $outlen$ is the bit length of the hash function output block.

The signature of a message M consists of the pair of numbers r and s that is computed according to the following equations:

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

z = the leftmost $\min(N, outlen)$ bits of $Hash(M)$.

$s = (k^{-1}(z + xr)) \bmod q.$

When computing s, the string z obtained from $Hash(M)$ shall be converted to an integer.

THE DIGITAL SIGNATURE ALGORITHM (DSA)

Note that r may be computed whenever k, p, q and g are available, e.g., whenever the domain parameters p, q and g are known, and k has been pre-computed, r may also be pre-computed, since knowledge of the message to be signed is not required for the computation of r. Pre-computed k, k^{-1} and r values shall be protected in the same manner as the the private key x until s has been computed.

The values of r and s shall be checked to determine if $r = 0$ or $s = 0$. If either $r = 0$ or $s = 0$, a new value of k shall be generated, and the signature shall be recalculated. It is extremely unlikely that $r = 0$ or $s = 0$ if signatures are generated properly.

The signature (r, s) may be transmitted along with the message to the verifier.

THE DIGITAL SIGNATURE ALGORITHM (DSA)

DSA signature verification and validation

Signature verification may be performed by any party (i.e., the signatory, the intended recipient or any other party) using the signatory's public key. A signatory may wish to verify that the computed signature is correct, perhaps before sending the signed message to the intended recipient. The intended recipient (or any other party) verifies the signature to determine its authenticity.

Let M', r', and s' be the received versions of M, r, and s, respectively; let y be the public key of the claimed signatory; and let N be the bit length of q. Also, let $\min(N, outlen)$ denote the minimum of the positive integers N and $outlen$, where $outlen$ is the bit length of the hash function output block.

The signature verification process is as follows:

1. The verifier shall check that $0 < r' < q$ and $0 < s' < q$; if either condition is violated, the signature shall be rejected as invalid.

THE DIGITAL SIGNATURE ALGORITHM (DSA)

2. If the two conditions in step 1 are satisfied, the verifier computes the following:

$w = (s')^{-1} \bmod q.$
z = the leftmost $\min(N, outlen)$ bits of $Hash(M')$.
$u1 = (zw) \bmod q.$
$u2 = ((r')w) \bmod q.$
$v = ((g^{u1} \cdot y^{u2}) \bmod p) \bmod q.$

The string z obtained from $Hash(M')$ shall be converted to an integer.

3. If $v = r'$, then the signature is verified. For a proof that $v = r'$ when $M' = M$, $r' = r$, and $s' = s$.

4. If v does not equal r', then the message or the signature may have been modified, there may have been an error in the signatory's generation process, or an imposter (who did not know the private key associated with the public key of the claimed signatory) may have attempted to forge the signature. The signature shall be considered invalid. No inference can be made as to whether the data is valid, only that when using the public key to verify the signature, the signature is incorrect for that data.

5. Prior to accepting the signature as valid, the verifier shall have assurances.

THANKS FOR ATTENTION