Message Authentication
&
Digital Signatures

Cryptography – CS 507
Erkay Savas
Sabanci University
erkays@sabanciuniv.edu
Outline

- Message Authentication Codes (MAC)
- Hash functions
- Digital signatures
Message Authentication Codes

- Message authentication ensures the integrity of message (i.e. its content has not been changed by unauthorized parties).

- Both message authentication and digital signatures can be viewed as having fundamentally two levels:
  - A function at the lower level which generates a value authenticator,
  - Authentication protocol that uses this function as primitive, enables a receiver to verify the authenticity of a message.
Message Authentication using Encryption

- The ciphertext of the entire message serves as its authenticator.
- Symmetric Encryption is used.
- If the receiver can decrypt the ciphertext into a meaningful plaintext, s/he can be sure that the message is authentic.
- There are several scenarios that encryption is not suitable:
  - It may be hard to distinguish a meaningful message.
  - Authentication cannot be done on selective basis.
  - Authentication cannot be done by only one destination on broadcast communication while the others are interested only in content
  - Separation of authentication and confidentiality may offer architectural flexibility
Message Authentication Codes (MAC)

- MAC or *cryptographic checksum* is a small fixed size block of data which is derived from an arbitrary length of message using a secret key.
- This technique assumes that two communicating parties, say A and B, share a secret key $K$.
- When A has a message $M$ to send to B, it calculates the MAC as function of the message and the key:
  \[
  \text{MAC} = C_K(M)
  \]
  where $C$ is the MAC function.
- The message plus MAC are transmitted to B.
Message Authentication Codes (MAC)

- B performs the same calculations on the received message, using $K$, to generate MAC to check if the attached MAC is the same.
- If an attacker alters the message it also needs to alter the MAC properly; this is something it cannot do without the knowledge of the secret key.
- A MAC function is similar to encryption in many ways.
- But it does not have to be reversible.
- A MAC function is a many-to-one function
Basic Uses of MAC

- Message authentication without confidentiality
Basic Uses of MAC

- Message authentication and confidentiality; authentication is tied to plaintext

Sender

Receiver
Basic Uses of MAC

Sender
- Message authentication and confidentiality; authentication is tied to ciphertext

Receiver

12/8/2002
Erkay Savas
Hash Functions

- One-way hash functions are similar to MAC; however, they do not use secret key.
- A hash function accepts an arbitrary length message $M$ and produces a fixed-length output, referred as hash code, or shortly hash, $H(M)$; message digest and hash value are also used.
- A change to any bit (or bits) in the message results in a change to the hash value.
- Hash functions are widely used in message authentication and digital signatures
Basic Use of Hash functions

\[ H(K(M)) = ? \]
Basic Use of Hash functions

\[ M \| K \rightarrow H \rightarrow D \]

Sender

Receiver

\[ M \rightarrow H \rightarrow =? \]

12/8/2002 Erkay Savas
Requirements for a hash function

- H can be applied to a block of data of any size
- H produces a fixed-length output.
- H(x) is relatively easy to compute for any given x, making both hardware and software implementations practical
- For any given value h, it is computationally infeasible to find x s.t. H(x) = h (one-way property)
- For any given block x, it is computationally infeasible to find y ≠ x s.t. H(x) = H(y) (weak-collision)
- It is computationally infeasible to find any pair (x, y) s.t. H(x) = H(y) (strong-collision)
Secure Hash Standard (SHA1)

- Proposed by NIST as standard hash function for certain US federal government applications.
- The hash value is 160-bits.
- Five 32-bit chaining variables are used.
- Similar to DES, the chaining variables are processed in 20 rounds.
- For more information see [http://www.nist.gov](http://www.nist.gov) and *Handbook of Applied Cryptography*. 
Birthday Paradox

- Probability results are sometimes counterintuitive.
- **The problem statement:** What is the minimum value of \( k \) s.t. the probability that at least two people in a group of \( k \) people have the same birthday is greater than 0.5. (Ignore the leap year).
- \( P(n, k) = \Pr \left[ \text{at least one duplicate in } k \text{ items, with each item able to take one of } n \text{ equally likely values between 1 and } n \right] \)
- In *birthday paradox*, we are looking for the smallest value of \( k \) s.t. \( (365, k) \geq 0.5 \).
- Assume \( n \geq k \).
Birthday Paradox

- Consider the number of different ways, $N$, that we can have $k$ values with no duplicates.
  
  \[ N = 365 \times 364 \times 363 \times \ldots (365-k+1) = \frac{(365!)}{(365-k)!} \]

- The total number of possibilities is $365^k$.

- Then the probability

  \[ P(365, k) = 1 - \frac{365!}{(365-k)!365^k} \]

- When the probabilities are calculated

  \[ P(365, 23) = 0.5073 \]
Birthday Paradox

• If there are 23 people in a room, the probability of two people having the same birthday is greater than 0.5.
• The probability is 89% that there is a match among 40 people.
• A useful inequality:

\[ P(n, k) > 1 - e^{-\frac{k(k-1)}{2n}} \]

• But it is not immediately straightforward to find a pair with the same birthday since, for example, in a group of 23, there are \((23(23-1))/2=253\) different pairs of people.
Birthday Attacks

- Suppose that $m=64$-bit hash code is used.
- An encrypted hash code $C = H(M)$ is transmitted with the corresponding unencrypted message $M$.
- An opponent would need to find an $M'$ s.t.
  \[ H(M) = H(M') \]
  to substitute another message and fool the receiver.
- Given a message $M$, on average an opponent would have to try about $2^{63}$ messages to find one that matches the hash value of the intercepted message.
- However, a different attack based on birthday paradox is much more feasible.
Birthday Attacks

• The opponent first forms two sets of messages:
  1. Generates $2^{m/2}$ variations on the original message, all of which conveys essentially the same meaning
  2. Prepares an equal number of messages, all of which are variations on the fake message to be substituted for the original message
• These two sets of messages are compared to find a pair of messages that produces the same hash value. The probability of success, by the birthday paradox, is greater than 0.5.
• If no match is found, additional messages are generated for the two sets.
Birthday Attacks

• The opponent offers the valid variation to the sender for signatures.
• The sender generates a hash value for this message and encrypts it.
• The opponent replace the original message with the fraudulent message that generates the same hash value.
• Attach the encrypted hash value to the fraudulent message and send it to the receiver.
• Since the hash value is 64-bit, the level of effort required is only on the order of \(2^{32}\).
Why Birthday Attacks Work?

- Variations are obtained by adding a space at the end of a line, modifying the punctuation, changing the wording slightly, etc.
- In two sets there are $k = 2^{m/2}$ messages each.
- The probability that a message from the first set of $k$ produces the same hash value as a message from the second set of $k$ is given by a similar formula with approximation

$$1 - e^{-k^2/n}$$
Why Birthday Attacks Work?

- $n = 2^{m/2} \Rightarrow$ Probability that there is a match between the hash values of two message from the two sets is approximately

$$1 - e^{-k^2/n} = 1 - e^{-1} = 0.63 > 0.5$$
Digital Signatures

- Digital signatures enables us to *personalize* electronic documents, i.e. to associate our identities to them.
- The assumption is that no one else can fake our signature for a given message.
- Why don’t we just digitize our *analog* signature and append it to a document?
- While classical signatures cannot be cut from a document and pasted into another document, the digital signatures can easily be forged.
- *We need digital signature that cannot be separated from a message and attached to another.*
Digital Signatures

- Digital signatures is not only tied to the signer but also to the message that is being signed.
- Digital signatures must be easily verified by the others.
- Therefore, digital signatures schemes consist of two distinct steps:
  1. The signing process
  2. The verification process
## RSA signatures

<table>
<thead>
<tr>
<th>Alice (signer)</th>
<th>Bob (verifier)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA Setup</td>
<td></td>
</tr>
<tr>
<td>(1) generates RSA public $(e_A, n)$ and private $(d_A, p, q)$ keys</td>
<td>(1) Receives $(y, m)$</td>
</tr>
<tr>
<td>(2) Alice’s signature for message $m$</td>
<td>(2) Downloads Alice’s public key $(e_A, n)$</td>
</tr>
<tr>
<td>$y \equiv m^{d_A} \mod n$</td>
<td>(3) Calculate</td>
</tr>
<tr>
<td></td>
<td>$z \equiv y^{e_A} \mod n$</td>
</tr>
<tr>
<td></td>
<td>and checks $z = m$</td>
</tr>
</tbody>
</table>
Blind signature with RSA

- Bob wants Alice to sign a document without her knowing its contents.
- The document contains a secret information (e.g. an invention by Bob) which Bob wants to be recorded publicly.
- The message to be signed by Alice is $m$.
- Alice uses RSA systems and tells her public key $(e, n)$ to Bob.
- Bob chooses a random integer $k < n$ with $\gcd(k, n) = 1$ and computes $t \equiv k^e m \pmod{n}$. He sends $t$ to Alice.
Blind signature with RSA

- Alice signs $t$ by computing $s \equiv t^d \pmod{n}$. She returns $s$ to Bob.
- Bob computes $s/k \pmod{n}$. This is the signed message $m^d \pmod{n}$ since
  
  $$s \div k \equiv t^d \div k \equiv (k^e m)^d \div k \equiv k^{ed} m^d \div k \equiv m^d \pmod{n}.$$ 

- $k$ is a random integer, so is $k^e \pmod{n}$. Therefore, $k^e m \pmod{n}$ gives no information about the message $m$. Alice knows nothing about what she is signing.
- Of course, Bob can trick Alice to sign a promise to pay him a million dollars. Safeguards are needed.
The ElGamal Signature Scheme

- With ElGamal signature scheme, there are many different signatures that are valid for a given message.
- Based on discrete logarithm.
- Alice chooses a large prime number $p$, a generator $\alpha$ in $\text{GF}(p)$.
- Alice next chooses a secret integer $a$ s.t. $1 \leq a \leq p-2$ and calculates $\beta \equiv \alpha ^a \pmod{p}$.
- $(\alpha, \beta, p)$ are public and $a$ is kept private.
- It is difficult for an adversary (Eve) to find $a$ given $\alpha$ and $\beta$ (discrete logarithm problem).
The ElGamal Signature Scheme

Alice *signs* the message *m* as in the following:

- Selects a random integer \( k \) s.t. \( \gcd(k, p-1) = 1 \).
- Computes \( r \equiv \alpha^k \pmod{p} \).
- Computes \( s \equiv k^{-1} (m-ar) \pmod{p-1} \).
- The signed message is the triple \((m, r, s)\)

Bob can *verify* the signature as follows:

- Downloads Alice’s public key \((\alpha, \beta, p)\)
- Compute \( v_1 \equiv \beta^r r^s \pmod{p} \) and \( v_2 \equiv \alpha^m \pmod{p} \)
- The signature is declared valid if and only if \( v_1 \equiv v_2 \pmod{p} \)
Why The ElGamal Verification Works?

- \( s \equiv k^{-1}(m-ar) \pmod{p-1} \Rightarrow m \equiv sk + ar \pmod{p-1} \)
- \( v_2 \equiv \alpha^m \pmod{p} \equiv \alpha^{sk+ar} \equiv (\alpha^a)^r (\alpha^k)^s \equiv \beta^r r^s \pmod{p} \).

\( \Rightarrow v_2 \equiv v_1 \pmod{p} \)

- If Eve (the opponent) knows private key \( a \), she can generate a valid signature for any given message.
- If she does not know \( a \) she cannot compute corresponding \( s \).
- Finding \( s \) given the following equation is equivalent to compute discrete logarithm:
  \( \beta^r r^s \equiv \alpha^m \pmod{p} \Rightarrow r^s \equiv \beta^{-r} \alpha^m \pmod{p} \Rightarrow s \log_\alpha r = -r \log_\alpha \beta + m \)
Session key must be unique

- Signer must choose a new and different session key $k$ for each message.
- If the same $k$ is used for two messages $m_1$ and $m_2$, then the same value of $r$ is used in both signatures.
- The $s$ values will be different, say $s_1$ and $s_2$.
- Eve knows that
  \[ s_1 k - m_1 \equiv -ar \equiv s_2 k - m_2 \pmod{p-1}. \]
- Therefore,
  \[ (s_1 - s_2)k \equiv m_1 - m_2 \pmod{p-1}. \]
  Let $d = \gcd(s_1 - s_2, p-1)$ then there are $d$ solutions to this congruence.
Session key must be unique

- Usually $d$ is small and so there are not many possible values of $k$.
- Eve computes $\alpha^k$ for each value of $k$ until she gets the value $r$ (recall that $r \equiv \alpha^k \pmod{p}$)
- She now knows $k$ and she can solve $ar \equiv m - s_k \pmod{p-1}$ for the private key $a$.
- There are $\gcd(r, p-1)$ possible solutions to $a$.
- Eve computes $\alpha^a$ for each value of $a$ until she gets the value $\beta$ since $\beta \equiv \alpha^a \pmod{p}$.
- She now has completely broken the system and can reproduce Alice’s signature at will.
Example: ElGamal Signature

- $m_1 = 151405$, $p = 225119$ and $\alpha = 11$ is a primitive root.
- $a$ is the private key and $\beta \equiv \alpha^a \equiv 18191 \pmod{p}$.
- To sign a message, Alice chooses a random integer $k$ and keeps it secret. She computes $\alpha^k \equiv 164130 \pmod{p}$.
  (nobody but she knows $a$ and $k$.)
- She computes $s_1 \equiv k^{-1}(m_1 - ar) \equiv 130777 \pmod{p-1}$. The signature triplet is $(151405, 164130, 130777)$
- Alice signs another message $m_2 = 202315$ and produces the signed message $(202315, 164130, 164889)$.
- Eve can immediately tell Alice uses the same $k$ for both messages.
Example: ElGamal Signature

- She therefore writes the congruence
  
  \[-314122k \equiv (s_1 - s_2)k \equiv m_1 - m_2 \equiv -50910 \pmod{p-1}.\]

  Since \( \gcd(-314122, p-1)=2 \) there are two solutions.

- \( k = \{239, 112798\} \).

- Since \( \alpha^{239} \pmod{p} \equiv 164130 = r, k = 239. \)

- Eve now writes
  
  \[s_1 \equiv k^{-1}(m_1-ar) \pmod{p-1}\]

  to obtain

  \[164310a \equiv ra \equiv m_1 - s_1k \equiv 187104 \pmod{p-1}.\]

  Since \( \gcd(164130, p-1) = 2 \), there are two solutions, namely \( a = \{28862, 141421\} \).

- Eve computes
  
  \[\alpha^{141421} \equiv 18191 \pmod{p} = \beta.\]

- Therefore \( a = 141421. \)
Signature with appendix

• The ElGamal signature scheme is an example of a signature with appendix.
• The message cannot be easily recovered from the signature \((r, s)\).
• The message \(m\) must be included in the verification process.
• RSA, in contrast to The ElGamal method, is a message recovery scheme.
The Digital Signature Algorithm (DSA)

- NIST proposed the DSA in 1991 and adopted it as a standard in 1994.
- It is similar to the ElGamal method since it is a signature with appendix scheme.
- It uses a hash value (message digest) that is signed.
- It utilizes SHA1 hash function which produces 160-bit hash values.
- We are trying to sign a 160-bit message.
The Digital Signature Algorithm (DSA)

Setup

- Alice finds a prime $q$ that is 160 bits long and chooses a prime $p$ that satisfies $q | p - 1$ ($q$ divides $p-1$).
- Let $g$ be primitive root mod $p$ and let $\alpha \equiv g^{(p-1)/q} \pmod{p}$ then $\alpha^q \equiv 1 \pmod{p}$.
- Alice chooses a secret $a$ s.t. $1 \leq a < q-1$ and calculates $\beta \equiv \alpha^a \pmod{p}$.
- Alice publishes $(p, q, \alpha, \beta)$.
- She keeps $a$ secret.
The Digital Signature Algorithm (DSA)

**Signing**
- Message $m$
- Computes $h = \text{SHA1}(m)$
- She selects a random, secret integer $k$ s.t. $1 \leq k < q-1$.
- Computes $r \equiv (\alpha^k \pmod{p}) \pmod{q}$.
- Computes $s \equiv k^{-1}(h + ar) \pmod{q}$.
- Alice’s signature for $m$ is $(r, s)$.
- Alice send $(r, s)$ and $m$ to Bob to verify.
The Digital Signature Algorithm (DSA)

**Verification**

- Bob downloads Alice’s public information \((p, q, \alpha, \beta)\).
- Computes \(h = \text{SHA1}(m)\)
- Computes \(u_1 \equiv s^{-1}h \pmod{q}\).
- Computes \(u_2 \equiv s^{-1}r \pmod{q}\).
- Computes \(v \equiv (\alpha^{u_1} \beta^{u_2} \pmod{p}) \pmod{q}\).
- Bob accepts the signature if and only if \(v = r\).

- Show that the verification really works.
Birthday attacks in DSA

- Fred is real estate agent and Alice wants to buy a land in Florida.
- She will sign a contract electronically using the DSA which necessitates her to sign the hash value of the contract.
- Suppose they use a hash function that produces 50-bit hashes instead of SHA1.
- Can Fred trick Alice to buy a swamp land in Florida while she thinks otherwise?
- Fred is unlikely to produce a fake contract which produces the hash value as the original contract.
- But he can use a different approach.
Birthday attacks in DSA

• Fred prepares the original contract for a nice piece of land.
• On the other hand, he also locates other places which have no value whatsoever.
• And he prepares $2^{30}$ different contracts for these junk lands by changing the wording slightly, placing a space at the end of a line, etc.
• He also prepares $2^{30}$ different variations of the original contract using the same tricks in which Alice does not notice the difference.
• He searches a match between the two sets of contracts which produces the same hash value.
Birthday attacks in DSA

- He is pretty sure he can find a match since the birthday paradox tells us the probability that there is a match when $k = 2^{30}$ and $n = 2^{50}$ is given by $1 - e^{-1024} \approx 1$.
- He gives this variation of the original contract to Alice to sign but appends the signature to the fake contract which produces the same hash value.