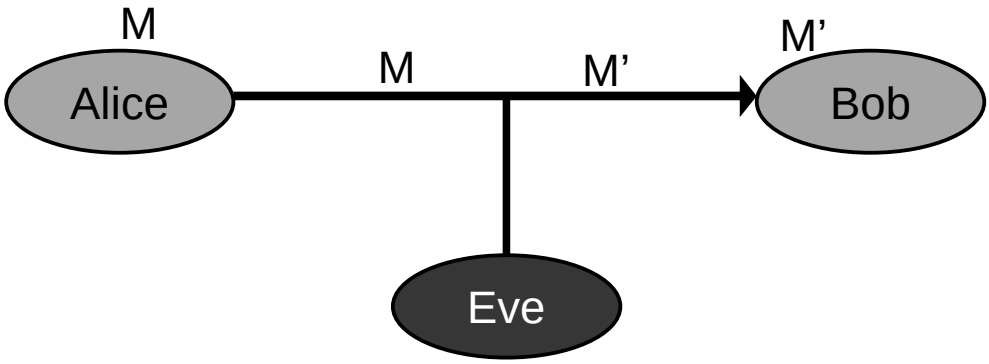
Introduction to Cryptography

Lecture 5

Benny Pinkas

Data Integrity, Message Authentication

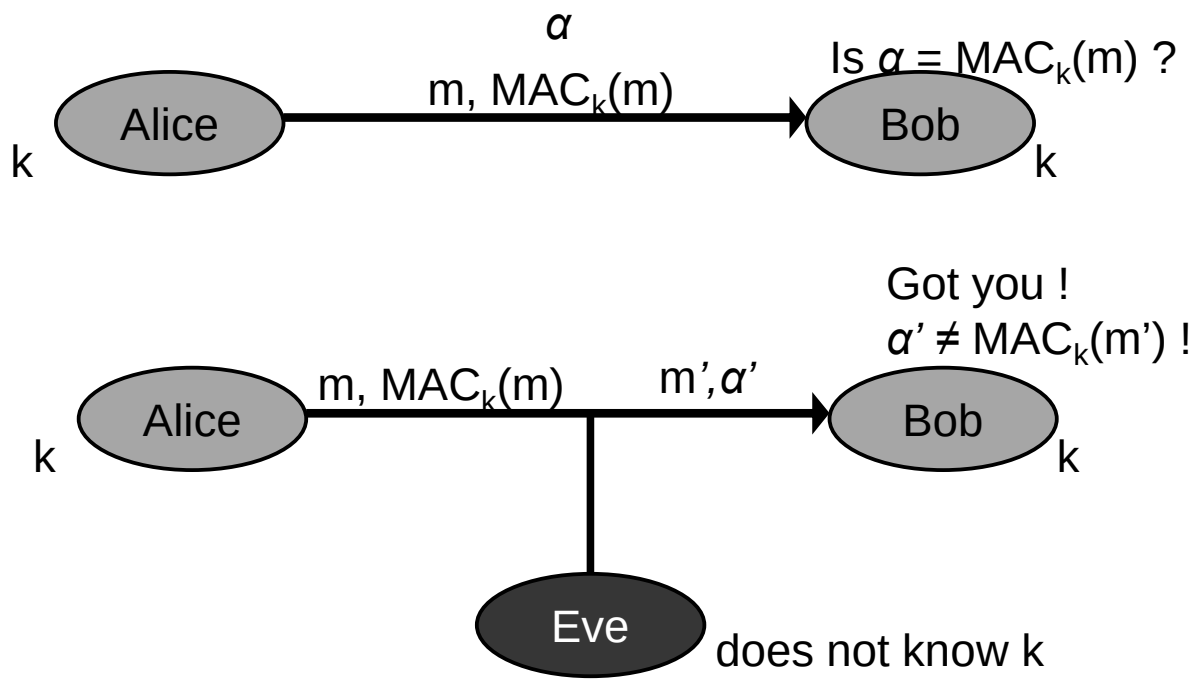
- Risk: an *active* adversary might change messages exchanged between Alice and Bob



• Authentication is orthogonal to secrecy. A relevant challenge regardless of whether encryption is applied.

• A one-time pad alone cannot prevent an adversary from changing the message.

Common Usage of MACs for message authentication



Requirements

- Security: The adversary,
  - Knows the MAC algorithm (but not K).
  - Is given many pairs $(m_i, MAC_K(m_i))$, where the m_i values might also be chosen by the adversary (chosen plaintext).
  - Cannot compute $(m, MAC_K(m))$ for any new m ($\forall i\ m \neq m_i$).
  - The adversary must not be able to compute $MAC_K(m)$ *even* for a message m which is "meaningless" (since we don't know the context of the attack).

- Efficiency: output must be of fixed length, and as short as possible.
  - $\Rightarrow$ The MAC function is not 1-to-1.
  - $\Rightarrow$ An n bit MAC can be broken with prob. of at least 2^{-n}.

Constructing MACs

- Based on block ciphers (CBC-MAC)

  or,

- Based on hash functions
  - More efficient
  - At the time, encryption technology was controlled (export restricted) and it was preferable to use other means when possible.

Hash functions

- MACs can be constructed based on hash functions.
- A hash function $h:X \rightarrow Y$ maps long inputs to fixed size outputs. ($|X|>|Y|$)
- No secret key. The hash function algorithm is public. (We will use it to construct MACs which use keys.)
- If $|X| > |Y|$ there are collisions *($x \neq x'$* for which *$h(x)=h(x')$)*.

Security definitions for hash functions

1. Weak collision resistance: for any $x \in X$, it is hard to find $x' \neq x$ such that $h(x)=h(x')$. (Also known as "universal one-way hash", or "*second* preimage resistance").
2. Strong collision resistance: it is hard to find any x,x' for which $h(x)=h(x')$.

- It's easier to find collisions. (Namely, under reasonable assumptions it holds that if it is possible to achieve (2) then it is also possible to achieve (1).) Therefore strong collision resistance is a stronger assumption.

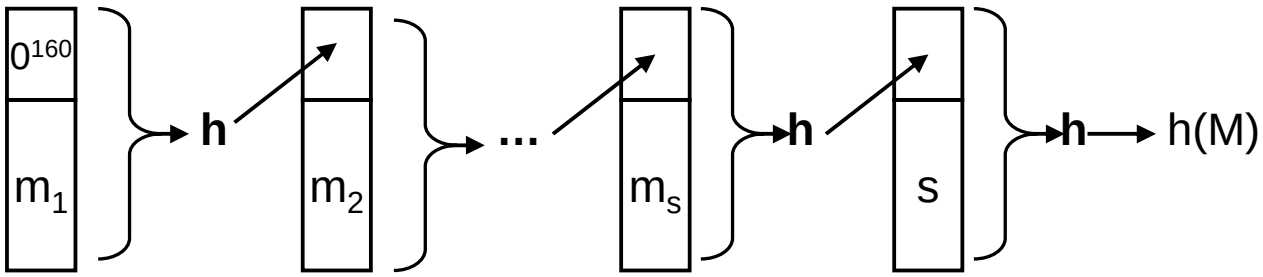
- Real world hash functions: MD5, SHA-1, SHA-256.

The Birthday Phenomenon (Paradox)

- For 23 people chosen at random, the probability that two of them have the same birthday is ½.
- Compare to: the prob. that one or more of them have the same birthday as Alan Turing is 23/365 (actually, $1-(1-1/365)^{23}$.)
  - More generally, for a random $h:X \rightarrow Z$, if we choose about $|Z|^{½}$ elements of X at random (1.17 $|Z|^{½}$ to be exact), the probability that two of them are mapped to the same image is > ½.
- Implication: it is harder to achieve strong collision resistance
  - A random function with an n bit output
    - Given y, can find $x \neq y$ s.t. $h(x)=h(y)$ after about 2^n tries.
    - Can find x,x' with $h(x)=h(x')$ after about $2^{n/2}$ tries.

From collision-resistance for fixed length inputs, to collision-resistance for arbitrary input lengths

- Hash function:
  - Input block length is usually 512 bits (|X|=512)
  - Output length is at least 160 bits (birthday attacks)
- Extending the domain to arbitrary inputs (Damgard-Merkle)
  - Suppose $h:\{0,1\}^{512} \rightarrow \{0,1\}^{160}$
  - Input: $M=m_1 \ldots m_s$, $|m_i|=512-160=352$. (what if $|M| \neq 352 \cdot i$ bits?)
  - Define: $y_0=0^{160}$. $y_i=h(y_{i-1},m_i)$. $y_{s+1}=h(y_s,s)$. $h(M)=y_{s+1}$.
  - Why is it secure? What about different length inputs?



Proof

- Show that if we can find M≠M' for which H(M)=H(M'), we can find blocks m ≠ m' for which h(m)=h(m').

- Case 1: suppose |M|=s, |M'|=s', and s ≠ s'
  - Then, collision: $H(M)=h(y_s,s) = h(y_{s'},s')=H(M')$
- Case 2: |M|=|M'|=s
  - Suppose that $H(M)=h(y_s,s)=h(y'_s,s)=H(M')$
  - If $y_s \neq y'_s$ then we found a collision in h.
  - Otherwise, go from i=s-1 to i=1:
    - $y_{i+1} = y'_{i+1}$ implies $h(y_i,m_{i+1}) = h(y'_I,m'_{i+1})$.
    - If $m_{i+1} \neq m'_{i+1}$, then we found a collision.
    - M ≠ M' and therefore there is an i for which $m_{i+1} \neq m'_{i+1}$

The implication of collisions

- Given a hash function with 2^n possible outputs. Collisions can be found
  - after a search of $2^{n/2}$ values
  - even faster if the function is weak (MD5, SHA-1)
- We find x, x' such that h(x)=h(x'), but we cannot control the value of x, x'.
- Can we find "meaningful" colliding values x, x' ?
  - The case of pdf files…

Basing MACs on Hash Functions

- Hash functions are not keyed. MAC_K uses a key.
- Best attack should not succeed with prob > $\max(2^{-|k|}, 2^{-|MAC()|})$.
- Idea: MAC combines message and a secret key, and hashes them with a collision resistant hash function.
  - E.g. $MAC_K(m) = h(k,m)$. (insecure.., given $MAC_K(m)$ can compute $MAC_K(m,|m|,m')$, if using the Damgard-Merkle construction)
  - $MAC_K(m) = h(m,k)$. (insecure.., regardless of key length, use a birthday attack to find m,m' such that h(m)=h(m').)

- How should security be proved?:
  - Show that if MAC is insecure than so is hash function h.
  - Insecurity of MAC: adversary can generate $MAC_K(m)$ without knowing k.
  - Insecurity of h: adversary finds collisions ($x \neq x'$, $h(x)=h(x')$.)

HMAC

- Input: message *m,* a key *K*, and a hash function *h*.
- $HMAC_K(m) = h(K \oplus opad, h(K \oplus ipad, m))$
  - where ipad, opad are 64 byte long fixed strings
  - K is 64 byte long (if shorter, append 0s to get 64 bytes).
- Overhead: the same as that of applying h to m, plus an additional invocation to a short string.

- It was proven [BCK] that if HMAC is broken then either
  - h is not collision resistant (even when the initial block is random and secret), or
  - The output of h is not “unpredcitable” (when the initial block is random and secret)
- HMAC is used everywhere (SSL, IPSec).

Basic Number Theory

Plan

- Basic number theory
  - Divisors, modular arithmetic
  - The GCD algorithm
  - Groups
- References:
  - Many books on number theory
  - Almost all books on cryptography
  - Cormen, Leiserson, Rivest, (Stein), “Introduction to Algorithms”, chapter on Number-Theoretic Algorithms.

Divisors, prime numbers

- We work over the integers
- A non-zero integer *b* divides an integer *a* if there exists an integer *c* s.t. $a=c \cdot b$.
  - Denoted as $b|a$
  - I.e. *b* divides *a* with no remainder
- Examples
  - Trivial divisors: $1|a,\ a|a$
  - Each of {1,2,3,4,6,8,12,24} divides 24
  - 5 does not divide 24
- Prime numbers
  - An integer *a* is prime if it is only divisible by 1 and by itself.
  - 23 is prime, 24 is not.

Modular Arithmetic

- Modular operator:
  - *a* mod *b,* (or *a%b*) is the remainder of *a* when divided by *b*
  - I.e., the smallest $r \geq 0$ s.t. $\exists$ integer *q* for which $a = qb+r$.
  - (Thm: there is a single choice for such *q,r*)

  - Examples
    - *12 mod 5 = 2*
    - 10 mod 5 = 0
    - -5 mod 5 = 0
    - -1 mod 5 = 4

Modular congruency

- *a* is congruent to *b* modulo *n* *(a ≡ b* mod *n)* if
  - *(a-b) = 0* mod *n*
  - Namely, *n* divides *a-b*
  - In other words, *(a* mod *n) = (b* mod *n)*
- E.g.,
  - *23 ≡ 12* mod *11*
  - *4 ≡ -1* mod *5*
- There are *n* equivalence classes modulo *n*
  - $[3]_7 = \{\ldots,-11,-4,3,10,17,\ldots\}$

Greatest Common Divisor (GCD)

- *d* is a common divisor of *a* and *b*, if *d*|*a* and *d*|*b.*
- *gcd(a,b)* (Greatest Common Divisor), is the largest integer that divides both *a* and *b*. *(a,b >= 0)*
  - *gcd(a,b)* = max *k s.t. k*|*a* and *k*|*b.*

- Examples:
  - gcd(30,24) = 6
  - gcd(30,23) = 1

- If *gcd(a,b)=1* then *a* and *b* are denoted relatively prime.

Facts about the GCD

- *gcd(a,b) = gcd(b, a* mod *b)* (interesting when a>b)
- Since (e.g., a=33, b=15)
  - If *c*|*a and c*|*b* then *c*|*(a mod b)*
  - If *c*|*b* and *c*|*(a mod b)* then *c*|*a*

- If *a* mod *b* = 0, then gcd(a,b)=b.

- Therefore,

gcd(19,8) =
gcd(8, 3) =
gcd(3,2) =
gcd(2,1) = 1

gcd(20,8) =
gcd(8, 4) = 4

Euclid's algorithm

Input: a>b>0

Output: gcd(a,b)

Algorithm:

1. if (*a mod b) = 0* return (*b*)
2. else return(gcd(*b, a* mod *b*))

Complexity:

- O(log *a*) rounds
- Each round requires $O(\log^2 a)$ bit operations
- Actually, the total overhead can be shown to be $O(\log^2 a)$

The extended gcd algorithm

Finding *s, t* such that $gcd(a,b) = a \cdot s + b \cdot t$

Extended-gcd(a,b) /* output is (gcd(a,b), s, t)

1. If (a mod b=0) then return(b,0,1)
2. (d',s',t') = Extended-gcd(b, a mod b)
3. (d,s,t) = (d', t', s'- $\lfloor a/b \rfloor \cdot t'$)
4. return(d,s,t)

Note that the overhead is as in the basic GCD algorithm

Groups

- Definition: a set G with a binary operation $\circ: G \times G \rightarrow G$ is called a group if:
  - (closure) $\forall a,b \in G$, it holds that $a \circ b \in G$.
  - (associativity) $\forall a,b,c \in G$, $(a \circ b) \circ c = a \circ (b \circ c)$.
  - (identity element) $\exists e \in G$, s.t. $\forall a \in G$ it holds that $a \circ e = a$.
  - (inverse element) $\forall a \in G \; \exists a^{-1} \in G$, s.t. $a \circ a^{-1} = e$.
- A group is Abelian (commutative) if $\forall a,b \in G$, it holds that $a \circ b = b \circ a$.

- Examples:
  - Integers under addition
    - (Z,+) = {…,-3,-2,-1,0,1,2,3,…}

More examples of groups

- Addition modulo N
  - $(G, \circ) = (\{0,1,2,\ldots,N-1\}, +)$

- Z_p^* Multiplication modulo a prime number p
  - $(G, \circ) = (\{1,2,\ldots,p-1\}, \times)$
  - E.g., $Z_7^* = (\{1,2,3,4,5,6\}, \times)$

- Trivial: closure (the result of the multiplication is never divisible by p), associativity, existence of identity element.
- The extended GCD algorithm shows that an inverse always exists:
  - $s \cdot a + t \cdot p = 1 \Rightarrow s \cdot a = 1 - t \cdot p \Rightarrow s \cdot a \equiv 1 \bmod p$

More examples of groups

- Z_N^* Multiplication modulo a composite number N
  - $(G, \circ) = (\{a \text{ s.t. } 1 \leq a \leq N-1 \text{ and } gcd(a,N)=1\}, \times)$
  - $E.g., Z_{10}^* = (\{1,3,7,9\}, \times)$
  - Closure:
    - $s \cdot a + t \cdot N = 1$
    - $s' \cdot b + t' \cdot N = 1$
    - $ss' \cdot (ab) + (sat' + s'bt + tt'N) \cdot N = 1$
  - Associativity: trivial
  - Existence of identity element: 1.
  - Inverse element: as in Z_p^*

Subgroups

- Let $(G, \circ)$ be a group.
  - $(H, \circ)$ is a subgroup of G if
    - $(H, \circ)$ is a group
    - $H \subseteq G$
  - For example, $H = (\{1,2,4\}, \times)$ is a subgroup of Z_7^*.

- *Lagrange's theorem:*
  If $(G, \circ)$ is finite and $(H, \circ)$ is a subgroup of $(G, \circ)$, then $|H|$ divides $|G|$

  For example: 3|6.

Cyclic Groups

- Exponentiation is repeated application of $\circ$
  - $a^3 = a \circ a \circ a.$
  - $a^0 = 1.$
  - $a^{-x} = (a^{-1})^x$
- A group G is cyclic if there exists a generator g, s.t. $\forall a \in G, \exists i$ s.t. $g^i = a$.
  - I.e., G= <g> = {1, g, g^2, g^3, …}
  - For example $Z_7^* = \langle 3 \rangle = \{1,3,2,6,4,5\}$
- Not all $a \in G$ are generators of G, but they all generate a subgroup of G.
  - E.g. 2 is not a generator of Z_7^*
- The order of a is the smallest $j>0$ s.t. $a^j = 1.$
- *Lagrange's theorem* $\Rightarrow$ for $x \in Z_p^*$, $ord(x) \mid p-1.$

Fermat's theorem

- Corollary of Lagrange's theorem: if $(G, \circ)$ is a finite group, then $\forall a \in G,\ a^{|G|}=1$.
- Corollary (Fermat's theorem): $\forall\, a \in Z_p^*,\ a^{p-1} = 1 \bmod p$. E.g., for all $\forall a \in Z_7^*$, $a^6=1$, $a^7=a$.
- Computing inverses:
- Given $a \in G$, how to compute a^{-1}?
  - Fermat's theorem: $a^{-1} = a^{|G|-1}$ $(= a^{p-2}$ in $Z_p^*)$
  - Or, using the extended gcd algorithm (for Z_p^* or Z_N^*):
    - $gcd(a,p) = 1$
    - $s \cdot a + t \cdot p = 1 \Rightarrow s \cdot a = -t \cdot p + 1 \Rightarrow$ s is a^{-1} !!
  - Which is more efficient?

Computing in Z_p^*

- P is a huge prime (1024 bits)
- Easy tasks (measured in bit operations):
  - Adding in O(log p) (linear n the length of p)
  - Multiplying in $O(\log^2 p)$ (and even in $O(\log^{1.7} p)$)
  - Inverting (a to a^{-1}) in $O(\log^2 p)$
  - Exponentiations:
    - $x^r \bmod p$ in $O(\log r \cdot \log^2 p)$, using repeated squaring