Foundations of Cryptography

CS579/CpE579—Spring 2011

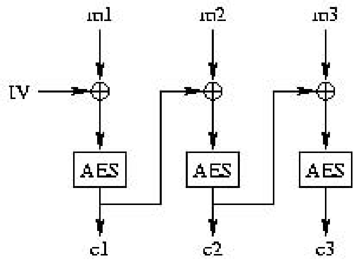
Instructor: Antonio R. Nicolosi　　　　Scribe: Vincent Lin

Lecture 10　　　　6 April 2011

1 Reverse Homework Review

1.1 Encryption and Message Authentication Codes (MACs)

In Lab 1 we have a system where we apply AES during CBC to encrypt the message, and then take the result and hash it with HMAC-SHA1:



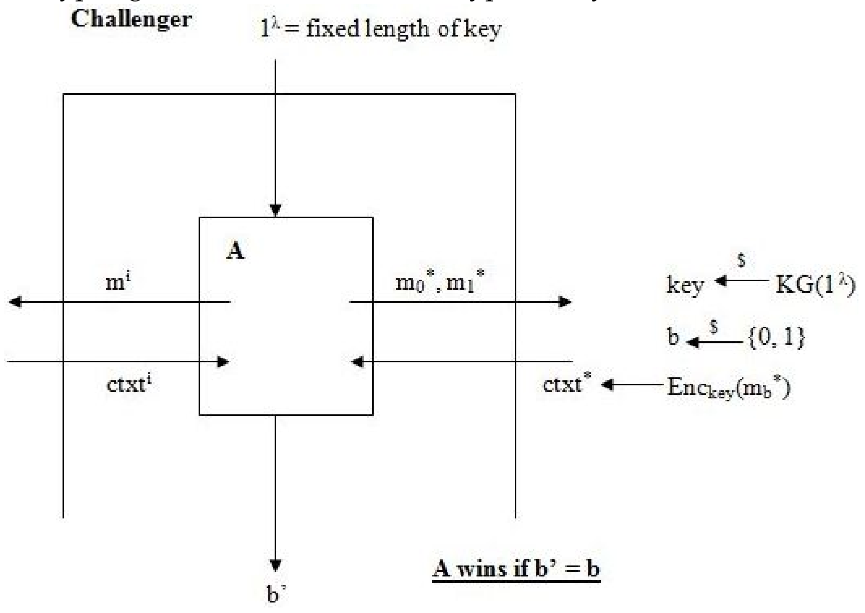
($c_0c_1c_2\ldots$ is then hashed with HMAC-SHA1 and the result is appended to the end of the encrypted file.)

But what if we do encryption/MAC in a different order?

- Encrypt then MAC (the method described above)
  This is what is used in Lab 1, and it always works while satisfying our definition of security.
- MAC then Encrypt
  This works most of the time, but there exists an issue with it that will be discussed later.
- Encrypt || MAC
  (Here we append the MAC of the original message to the end of the encryption of that same message.) This method is broken because it does not satisfy our definition of security.

1.2 Definition of Security

Building upon our original definition of security, we can make the game easier for the attacker by giving access to an encryption oracle that takes an input m^i and returns the ciphertext created by encrypting m^i with the selected encryption key:


Challenger
1^λ = fixed length of key
A
m^i
$ctxt^i$
m_0^*, m_1^*
$ctxt^*$
$key \xleftarrow{\$} KG(1^\lambda)$
$b \xleftarrow{\$} \{0, 1\}$
$ctxt^* \leftarrow Enc_{key}(m_b^*)$
b'
A wins if b' = b

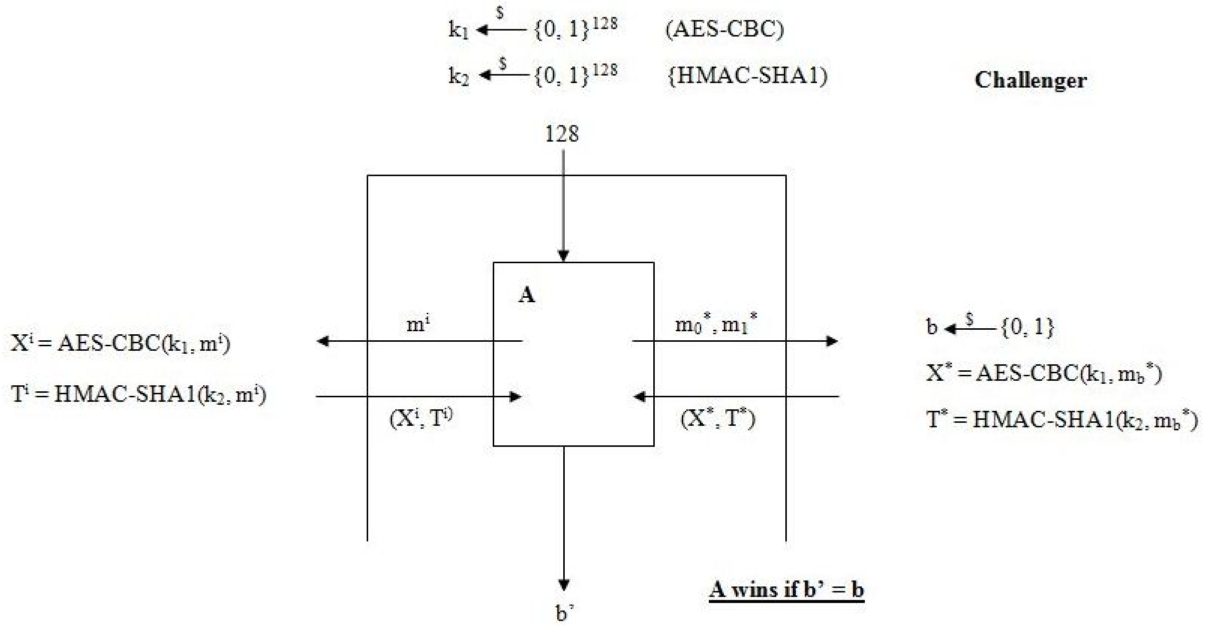
More formally, ∀ efficient attackers A,

$\exists\, \gamma$ s.t. Pr[b = b′ | k ← KG(1^λ), b ← {0, 1}, b′ ← $A^{Enc(k,\cdot)}$, Enc(k, select(b,·,·))] $\leq \frac{1}{2} + \gamma$

(*It should be noted that this updated game will never work with a deterministic encryption scheme. In general, a secure encryption scheme can never be deterministic.)

1.3 Encrypt || MAC

Now consider Encrypting using AES-CBC and hashing with HMAC-SHA1:


$k_1 \xleftarrow{\$} \{0, 1\}^{128}$ (AES-CBC)
$k_2 \xleftarrow{\$} \{0, 1\}^{128}$ {HMAC-SHA1)
Challenger
128
A
m^i
X^i = AES-CBC(k_1, m^i)
T^i = HMAC-SHA1(k_2, m^i)
(X^i, T^i)
m_0^*, m_1^*
(X^*, T^*)
$b \xleftarrow{\$} \{0, 1\}$
X^* = AES-CBC(k_1, m_b^*)
T^* = HMAC-SHA1(k_2, m_b^*)
b'
A wins if b' = b

Since HMAC-SHA1 is deterministic by construction, the second portion of the ciphertext is generated deterministically and the scheme is broken. An attacker can just compare T's.

MACs are used for data integrity and authentication. If you want secrecy, you need encryption, and if you want to combine them, you should encrypt the message first and then use the ciphertext for appended MACs instead of using the plaintext.

1.4 MAC then Encrypt

The one caveat for MAC then Encrypt has to do with message length. If we do not specify that $| m_0^* | = | m_1^* |$ then the scheme has a problem. Encryption does not guarantee the protection of the size of the file. If one message is 1 byte and the other is 1 terrabyte, the ciphertext sizes could be very different in size. This does not matter for our library because HMAC-SHA1 takes a message of any length up to 2^{64} bytes and hashes it to a 20-byte output. This guarantees that all messages being encrypted will be the same length, but this may not be the case with other MAC protocols.

There is no requirement that forces MACs to always generate the same sized output from any input, and it is also okay for messages of the same length to have MACs of different length.
e.g. compression - Some files do not compress as well as others (think text files vs images) so 2 files of the same length originally might compress to different lengths. So compressing and then MACing might cause the MACs generated to have different lengths before you encrypt the files, and result in ciphertexts of very different lengths.

2 Modulo Group Theory

2.1 Review

($\mathbb{Z}_n$, +) always works as an additive group:

- $\mathbb{Z}_n$ is closed under the group operation (addition):
  $\forall$ a, b $\in \mathbb{Z}_n$, (a + b) $\in \mathbb{Z}_n$.
- The identity element, e, exists:
  $\forall$ a $\in \mathbb{Z}_n$, a = a + 0 = 0 + a = a.
- ($\mathbb{Z}_n$, +) has the associative property:
  $\forall$ a, b, c $\in \mathbb{Z}_n$, (a + b) + c = a + (b + c).
- All elements of $\mathbb{Z}_n$ have an inverse:
  $\forall$ a $\in \mathbb{Z}_n$, $\exists\, a^{-1}$ s.t. e = a + a^{-1} = a^{-1} + a = e

However this is not the case under multiplication. For example, ($\mathbb{Z}_{26}$, $\cdot$) is closed, has an identity element (1), and is associative, but not every element in ($\mathbb{Z}_{26}$, $\cdot$) has an inverse:
Inverses: $\forall$ a $\in \mathbb{Z}_{26}$ $\exists\, a^{-1}$ s.t. a $\cdot$ a^{-1} = 1 mod 26 $\Longleftrightarrow$ gcd(a, 26) = 1

gcd(2, 26) = 2 $\neq$ 1 $\Longrightarrow$ 2 does not have an inverse in ($\mathbb{Z}_{26}$, $\cdot$) $\Longrightarrow$ ($\mathbb{Z}_{26}$, $\cdot$) is not a group

2.2 Compute Inverse Mod(a, n)

This algorithm takes n and a, and returns the inverse of a in $\mathbb{Z}_n$:

```
    $x_{old}$ = $n$, $x_{curr}$ = $a$, $x_{new}$
    $s_{old}$ = 1, $s_{curr}$ = 0, $s_{new}$
    $t_{old}$ = 0, $t_{curr}$ = 1, $t_{new}$

    while $x_{curr} > 0$
        $q = \lfloor x_{old}/x_{curr} \rfloor$
        $x_{new} = x_{old} - q*(x_{curr})$
        $s_{new} = s_{old} - q*(s_{curr})$
        $t_{new} = t_{old} - q*(t_{curr})$
    done
    assert $x_{curr} == 1$
    return $s_{curr}$
```

(As mentioned in class, when only looking for the inverse of a, t is not used in the algorithm, but it is necessary in order to prove that this algorithm outputs the correct values.)

2.3 $\mathbb{Z}_n^*$

Now we continue with the issue from 1.1 where $\mathbb{Z}_{26}$ is not a group because gcd(2, 26) = 2 $\neq$ 1. In order to satisfy the Inverses axiom, all elements of the set must be relatively prime to 26 (gcd(a, 26) = 1). Now consider a set $\mathbb{Z}_{26}^*$ defined as follows:

$\mathbb{Z}_{26}^* := \{a \in \mathbb{Z}_{26} \text{ s.t. } \gcd(a, 26) = 1\}$

(This is essentially all of the numbers in $\mathbb{Z}_{26}$ that are relatively prime with 26. Because they are relatively prime, they must also have inverses)

2.3.1 Is ($\mathbb{Z}_{26}^*$, $\cdot$) a group?

- *Identity*: 1 is inherited from ($\mathbb{Z}_{26}$, $\cdot$) as the identity.
- *Associativity*: this is inherited from ($\mathbb{Z}_{26}$, $\cdot$).
- *Existence of Inverses*: Everything must have an inverse since we constructed $\mathbb{Z}_{26}^*$ specifically so that all elements would have an inverse.
- *Closure*: This is the only non-trivial one, but since gcd(a mod n, n) = 1 & gcd(b mod n, n = 1) $\implies$ gcd(a $\cdot$ b mod n, n) = 1, we know that $\mathbb{Z}_{26}^*$ is closed under multiplication.

Yes ($\mathbb{Z}_{26}^*$, $\cdot$) is a group, and we can generalize this to ($\mathbb{Z}_n$, $\cdot$)

2.3.2 What is the size of $\mathbb{Z}_n^*$?

$|\mathbb{Z}_n| = n$

$|\mathbb{Z}_n^*| < n \rightarrow |\mathbb{Z}_n^*| < n - p - q + 1$ if n is a composite number where n = pq (p, q prime and p $\neq$ q):

$n - p - q + 1 = pq - p - q + 1 = p(q - 1) - (q - 1) = (p - 1)(q - 1) = |\mathbb{Z}_p^*| \cdot |\mathbb{Z}_q^*|$

$|\mathbb{Z}_p^*| = p - 1$ if p is prime (everything in $\mathbb{Z}_n$ is coprime to p, and we do not include 0)

p prime $\implies \gcd(a, p) = 1 \;\forall\; a \in \mathbb{Z}_p \backslash \{0\}$

2.4 Euler Totient Function

The Euler Phi function, as it is also refered to, is used to more easily compute the number of elements in $\mathbb{Z}_n^*$ (ϕ(n) = $|\mathbb{Z}_n^*|$).

We know that ϕ(p) = $|\mathbb{Z}_p^*|$ = p - 1 (p prime).
This also gives us ϕ(p·q)= ϕ(p)·ϕ(q) (p, q prime and p $\neq$ q)
For general n, $\phi(p_1^{e_1} \cdot \ldots \cdot p_r^{e_r})$ = $\phi(p_1^{e_1}) \cdot \ldots \cdot \phi(p_r^{e_r})$ = $p_1^{e_1-1}(p_1$ - 1)$\cdot \ldots \cdot p_r^{e_r-1}(p_r$ - r)

Knowing the factorization of n is essentially like knowing the order of $\mathbb{Z}_n^*$

2.5 Prime Numbers

2.5.1 Density of primes in $\mathbb{Z}$

Specifically, in the range 0 to 2^k, what is the total number of primes?
The number of primes is roughly $\theta(\frac{2^k}{k}) \Longrightarrow$ in the range 0 to m, the number of primes is roughly $\theta(\frac{m}{log(m)})$

We can now use this to find the expected number of iterations that it would take to select a random number from 0 to k and check if it is prime:
Pr[prime] = $\frac{\frac{2^k}{k}}{2^k}$ = $\frac{1}{k} \Longrightarrow$ E(prime) = $\frac{1}{\frac{1}{k}}$ = k

2.5.2 Testing the primality of r

```
for i ← 1 to $\sqrt{r}$ do
    if $\frac{r}{i}$ is an integer (r mod i == 0)
        no
yes
```

This is a very naive algorithm to simply test all possible values up to $\sqrt{r}$ to see if r is a prime. Say r = 512 bits (r $\approx 2^{512}$). The loop runs for $\sqrt{2^{512}}$ = 2^{256}. This is not polynomial time, and the run time is O($2^{\frac{k}{2}}$).

Other tests:

- *Miller-Rabin*: O(k^2) (Probabilistic polynomial time)
  Returns either 'yes it is composite' or 'I don't know'
- *AKS (2002)*: O($k^1 1$)
  This test does go through all possible values, and is deterministic. It has been improved since its development to O(k^6), however running Miller-Rabin multiple times is still generally preferred.

2.6 Cyclic Groups

2.6.1 Fermat's Little Theorem

For p prime,

$\forall$ a $\in \mathbb{Z}_p^*$ $\quad a^{p-1} \cong 1$ mod p.

2.6.2 Definition of a Cyclic Group

A cyclic group is defined as a group where all elements can be represented as powers of a single element, called a generator (sometimes denoted with g).

More formally, a group, G, is said to be cylic if $\exists$ a $\in$ G s.t. G = $\{1, a, a^2, ..., a^{\phi(n)-1}\}$

2.6.3 Examples of Cyclic Groups

($\mathbb{Z}_n$, +) is always a cyclic group because you can simply take 1 and add 1 repeatedly to generate all elements of the set $\mathbb{Z}_n$.

($\mathbb{Z}_7^*$, $\cdot$) is also a cyclic group:
($\mathbb{Z}_7^*$, $\cdot$) = $\{1,2,3,4,5,6\}$
Now lets try to find a generator, we will test 2.
2^0mod(7)$\cong$1, 2^1mod(7)$\cong$2, 2^2mod(7)$\cong$4, 2^3mod(7)$\cong$1, ⊠
Now 3,
3^0mod(7)$\cong$1, 3^1mod(7)$\cong$3, 3^2mod(7)$\cong$2, 3^3mod(7)$\cong$6,
3^4mod(7)$\cong$4, 3^5mod(7)$\cong$5, 3^6mod(7)$\cong$1, ✓

2.6.4 The Order of $\mathbb{Z}_n^*$ and g

(As mentioned above, g is called a generator or a primitive root).
G = ($\mathbb{Z}_p^*$, $\cdot$) is a cyclic group $\iff \exists$ g $\in \mathbb{Z}_p^*$ s.t. $\mathbb{Z}_p^*$ = $\{1, g, g^2, g^3, \ldots\}$

Since $\mathbb{Z}_p^*$ is finite while the powers of g can go up infinitely, we know that the powers of g must wrap around at some point (pidgeonhole principle).
e.g g^r mod p = 1 where r is called the order of the element g. This is denoted as ord(g) = r
So from our 2 examples in the previous section, ord(2) in ($\mathbb{Z}_7^*$, $\cdot$) is 3 and ord(3) is 6.

The order of G = ($\mathbb{Z}_p^*$, $\cdot$) is the order of its generator $\Rightarrow$ the order of $\mathbb{Z}_p^*$ is p - 1. This can also be denoted as |G| = p - 1

Going back to Fermat's Little Theorem, if an element is in $\mathbb{Z}_p^*$ and p prime, its order is always p - 1 or a factor of p - 1.

2.6.5 Lagrange's Theorem

If H is a subgroup of G (H $<$ G) then $\frac{|H|}{|G|}$ (the order of H divides the order of G)
e.g. H = $\{1, 2, 4\} < (\mathbb{Z}_7^*, \cdot) = \{1, 2, 3, 4, 5, 6\}$ = G

H is closed:

```
  | 1 | 2 | 4 |
----------------
1 | 1 | 2 | 4 |
----------------
2 | 2 | 4 | 1 |
----------------
4 | 4 | 1 | 2 | (the multiplication table for {1, 2, 4}
```

1 is the identity element, and all elements have an inverse (1's inverse is itself, and 2 and 4 are the inverses of one another). Associativity is inherited from G so H is a group itself as well as a subgroup of G.
$|G| = |\mathbb{Z}_7^*|$ = p - 1 = 6
$|H| = |\{1, 2, 4\}|$ = 3
$3|6$ ✓

$<$a$>$ = $\{a, a^2 \bmod p, \ldots, a^r \bmod p, \ldots\}$ ($<$a$>$ denotes that a is a generator)

By closure, we get that: $(a^i \bmod p) \cdot (a^j \bmod p) = a^{i+j} \bmod p$

o = ord(a)|(p - 1)
p - 1 = o $\cdot$ c (for some constant c)
$\Rightarrow a^o \bmod p = 1$ (by the definition of the order of an element)
$= a^{o \cdot c} = 1^c = 1 \bmod p$
$= a^{p-1} = 1 \bmod p$ (but this does not include 0 since $0 \notin \mathbb{Z}_p^*$)
$= a^p \cdot a^{-1} = 1 \bmod p$ (using the property gained from closure)
$\Longrightarrow a^p \cdot a^{-1} \cdot a = (1 \bmod p) \cdot a$
$\Longrightarrow a^p = a \bmod p$

To generalize: $\forall\, a \in \mathbb{Z}_p, \qquad a^p \cong a \bmod p$

2.7 Discrete Logarithms

We already know that 3 is a generator of $\mathbb{Z}_7^*$.

$<3> = \{1, 3, 2, 6, 4, 5\}$

$(1 = 3^0 \text{ mod } 7)$ $(3 = 3^1 \text{ mod } 7)$ $(2 = 3^2 \text{ mod } 7)$

$(6 = 3^3 \text{ mod } 7)$ $(4 = 3^4 \text{ mod } 7)$ $(5 = 3^5 \text{ mod } 7)$

Now consider the following function *f(x)*: $\mathbb{Z}_{p-1} \to \mathbb{Z}_p^*$

e.g. $\mathbb{Z}_6 \to \mathbb{Z}_7^*$

x $|\!\to 3^x$ mod 7

$0 \to 1$

$1 \to 3$

$2 \to 2$

$3 \to 6$

$4 \to 4$

$5 \to 5$

f is a bijection and raising a base to an exponent over a modulus is called modular exponentiation.

The inverse function of f is:

$\mathbb{Z}_p^* \to \mathbb{Z}_{p-1}$

y $|\!\to$ s.t. 3^x = y mod 7

This function is called the Discrete Logarithm (Dlog.

e.g $\quad Dlog_3\, 6 = 3 \Longrightarrow 3^3 \text{ mod } 7 = 6$ ✓

2.7.1 Efficiency

For large numbers, g^x mod p, you don't want to compute g^x and then reduce:

e.g. $|g|$ = 1024 bits (2^{1024})

e.g. $|x|$ = 160 bits (2^{160})

e.g. $|p|$ = 1024 bits (2^{1024})

$g^x = (2^{1024})^{2^{160}} \sim (2^{2^{10}})^{2^{160}} \sim 2^{2^{170}} \sim 2^{170}$ (too many steps)

Possible solution:

Try $\quad g \quad g^2 \quad g^3 \quad \ldots \quad g^x$

and do (mod p) at each step so that you don't grow in size. Unfortunately this doesn't work because there are too many steps to take (x steps). In the above example, $|x|$ = 160 bits $\Longrightarrow \approx 2^{160}$ steps. $\Longrightarrow$ too many steps.

2.7.2 Horner's Rules of Multiplication

x : 160-bits
$(x_{159}x_{158}\ldots x_1x_0)_2$ = x (binary representation of x)
$\Longrightarrow$ x = $x_0 + x_1 2^1 + x_2 2^2 + \ldots + x_{159}2^{159}$

$$= \sum_{i=0}^{159} x_i 2^i$$

To compute modular computation:

$$g^x = g^{x_0 + x_1 2^1 + \ldots + x_{159}2^{159}}$$
$$= g^{x_0} g^{2(x_1 + \ldots + x_{159}2^{159})}$$
$$= g^{x_0} g^{(x_1 + \ldots + x_{159}2^{159})2}$$

So now consider a recursive algorithm to calculate modular exponentiation called ModExp:
ModExp(g, p, x, k) (where k = log(x))

O(k)	ModExp k - 1 bits
O(k^2)	squaring mod p
O(k^2)	multiplication mod p

Square + Multiply algorithm gives us a O(k^3) run time to calculate modular exponentiate, which is efficient.

References

[1] Lecture Notes. 'CS579 Foundations of Cryptography'. Antonio Nicolosi. April 6. 2011.

[2] Scribe Notes. 'CS579 Foundations of Cryptography'. Joe Geis. April 14. 2010.