# CO 487 – Applied Cryptography.

## 1. Overview of Cryptography.

The fundamental goals of cryptography are as follows:

- *Confidentiality*: Keeping data secret from all but those authorized to see it
- *Data integrity*: Ensuring data has not been altered by unauthorized means
- *Data origin authentication*: Confirming the source of the data
- *Non-repudiation*: Preventing an entity from denying previous commitments or actions

*Information security* describes all measures used to protect information from deliberate or inadvertent unauthorized acquisition, damage, disclosure, manipulation, modification, or loss.

It is a challenge to build high-confidence systems that give reasonable assurance of security, particularly when it is not valued monetarily for most purposes.

Information security includes models of computer, network, and software security; cryptography is just a mathematical tool that assists in the world of information security. It is, however, essential.

## 2. Symmetric Key Encryption Schemes.

**Definition.** A *symmetric-key encryption scheme* (SKES) consists of the plaintext space $M$, the ciphertext space $C$, the keyspace $K$, a family of encryption functions $E_k \colon M \to C,\ \forall\ k \in K$, and a family of decryption functions $D_k \colon C \to M,\ \forall\ k \in K$ such that

$$D_k(E_k(m)) = m\ \forall\ k \in K \text{ and } E_k(m) = c \in C.$$

To use a SKES,

1. Alice and Bob agree on a secret key $k \in K$ by communicating over a secure channel.
2. Alice computes $c = E_k(m)$ and sends Bob the ciphertext $c$ over the unsecured channel.
3. Bob can retrieve the plaintext by computing $m = D_k(c)$.

The *substitution cipher* uses all English messages as the message space and all permutations of the English alphabet as the keyspace. The encryption function applies the permutation given in the key to each letter of the message, and the decryption function applies the inverse permutation.

But this isn't very secure... people break these in puzzles in the newspapers!

There exist many kinds of generic attacks for encryption schemes. They can generally be divided as follows:

- Passive attacks
  - *Ciphertext-only attack*: The attacker tries to derive information by simply viewing ciphertexts.
  - *Known-plaintext attack*: The attacker knows some plaintext and the corresponding ciphertext.
- Active attacks
  - *Chosen-plaintext attack*: The attacker chooses some plaintext and sends it to Alice to obtain the corresponding ciphertext.
  - *Chosen-ciphertext attack*: The attacker has a target ciphertext to decrypt, and can obtain the plaintext, ciphertext pair for any other message.

- Other attacks
  - *Side-channel attacks*: The attacker monitors the encryption and decryption equipment to try to learn about the scheme.
  - *Clandestine attacks*: Bribery, blackmail, $5 wrench, etc.

In addition to the adversary's attack capabilities, she may also be limited computationally:

- *Information-theoretic security*: The attacker has infinite computational resources.
- *Complexity-theoretic security*: The attacker has a 'polynomial-time Turing machine.'
- *Computational security*: The attacker is computationally bounded by their resources.

What is the adversary's goal? There could be varying degrees of what she is trying to achieve.

1. Recover the secret key.
2. Systematically recover plaintext from ciphertext, not necessarily learning the secret key.
3. Learn some partial information about the plaintext from the ciphertext, other than its length.

If the adversary can achieve 1 or 2, the scheme is said to be *totally insecure*. If the adversary cannot achieve 3, the scheme is said to be *semantically secure*.

**Definition.** A SKES is said to be *secure* if it is semantically secure against chosen-plaintext attacks by a computationally bounded adversary.

In general, we want symmetric-key encryption schemes to have efficient encryption/decryption algorithms, a small key size (large enough to protect from exhaustive key search), and they should be secure, even against the designer of the system.

The simple substitution cipher is totally insecure—apply the chosen-plaintext attack using the string "A..Z". In fact, it is also totally insecure against ciphertext-only attacks. Obtaining just a few hundred characters allows for a letter distribution analysis, which can be used to determine the secret key.

However, exhaustive keysearch is impossible. There are $26! \approx 2^{88}$ permutations of the English alphabet.

**Computational Feasibility.**

For today's (2012) computing power, we define the following classifications of computational difficulty.

- $2^{40}$ operations is considered *easy*.
- $2^{56}$ operations is considered *feasible*.
- $2^{64}$ operations is considered *barely feasible*.
- $2^{80}$ operations is considered *infeasible*.
- $2^{128}$ operations is considered *totally infeasible*.

The *one-time pad* is another SKES, that can be proved to be semantically secure against ciphertext-only attacks by an adversary with infinite computational resources. However, the key size required for this "perfect secrecy" is impractically large, and so the one-time pad is unpopular and impractical for general use.

## 3. Stream Ciphers.

What if a seemingly random stream of bits could be used to simulate the one-time pad, while still using a small key? This is the idea behind stream ciphers.

We generate a pseudorandom (rather than purely random) *keystream* from a *seed*. The seed is the secret key shared between Alice and Bob.

For the pseudorandom bit generator to be secure, it must produce a keystream indistinguishable from random bits (*indistinguishability requirement*), and it should be infeasible to learn any information about the rest of the keystream given small portions of it (*unpredictability requirement*).

The *RC4* stream cipher is one of the most widely used in practice today. It's fast, it's simple, it has variable key sizes, and no significant weaknesses for it have been found. It generates a "random" permutation of 0-255 from the secret key, and then uses this to generate the keystream.

One of the applications of RC4 is in the WEP security standard. WEP had three goals:

- *Confidentiality*: Prevent eavesdropping.
- *Data integrity*: Prevent tampering with transmitted messages.
- *Access control*: Protect access to the wireless network infrastructure.

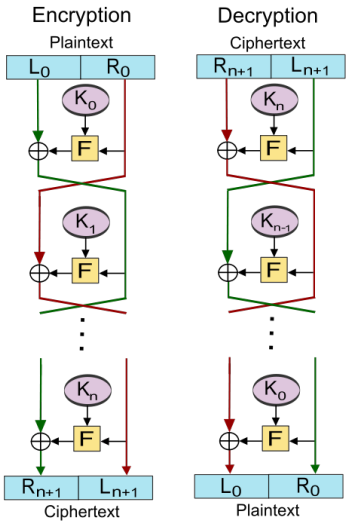
However, it fails in all these goals. WPA2 was designed to replace WEP and fix its security issues by using the AES block cipher and deal with the IV issues.

## 4. Block Ciphers, Feistel Ciphers.

A *block cipher* is a SKES that breaks the plaintext into blocks of fixed length, and encrypts the blocks one at a time. The stream cipher, on the other hand, encrypted the plaintext one character (a bit) at a time.

For a good block cipher, we want a small key (but large enough to avoid exhaustive key search), a complex relationship between key and ciphertext bits, and each ciphertext bit to depend on all the plaintext bits in the message. It is also desirable for the encryption and decryption rates to be fast.

The *Feistel Cipher*.



**Meet in the Middle Attack on Double-DES.**

Given three plaintext/ciphertext pairs, it is computationally feasible with high probability to recover the secret key for double-DES $(k_1, k_2)$.

## 5. Hash Functions.

A *hash function* is a mapping $H\colon \{0,1\}^* \to \{0,1\}^n$, where $H(x)$ can be efficiently computed for all $x \in \{0,1\}^*$.

We require hash functions to have the following desirable properties:

- *Preimage Resistance*: Given a random hash value $y \in \{0,1\}^n$, it is computationally infeasible to find (with non-negligable probability of success) any input $x$ such that $H(x) = y$.
- *2nd Preimage Resistance*: Given a random input $x \in \{0,1\}^*$, it is computationally infeasible to find (with non-negligable probability of success) a second input $y \neq x$ such that $H(y) = H(x)$.
- *Collision Resistance*: It is computationally infeasible to find (with non-negligable probability of success) two distinct inputs $x$, $y$, such that $x \neq y$ and $H(x) = H(y)$.

Collision resistance implies 2nd preimage resistance, because if it is computationally infeasible to find any two distinct inputs that hash to the same value, it will be computationally infeasible to find something that hashes to the same hash value as another value.

However, the converse is not true. Just because it is hard to find something that hashes to the same value for a particular value, doesn't mean that it is in general difficult to find two arbitrary values that hash to the same value.

Though collision resistance does not guarantee preimage resistance, in practice, so long as $H$ hashes uniformly, then collision resistance does imply preimage resistance.

**Van Oorschot and Weiner Generic Collision Search.**

A clever collision-finding attack that efficiently finds collisions with very small space requirements. Additionally, it is easily parallelizable.

**Iterated Hash Functions.**

They begin with an initial value, and use a compression function $f$ to hash and combine the value to encrypt in a chained manner.

By Merkle's theorem, if $f$ is secure, then the iterated hash function using $f$, $H$, is secure.

## 6. MAC Schemes.

A *message authentication code* (MAC) scheme is a family of functions $H_k\colon \{0,1\}^* \to \{0,1\}^n$ parameterized by an $\ell$-bit key $k$, where each function $H_k$ can be efficiently computed.

They are used for providing data integrity and data origin authentication. However, non-repudiation is, for the most part, impractical. If a trusted third party is used, this may be possible.

**Definition.** We say that a MAC scheme is *secure* if given some MAC tag pairs $(x_i, H_k(x_i))$, it is computationally infeasible to compute (with non-negligable probability of success) a pair $(x, H_k(x))$ for any new message $x$.

**Attacks on Cipher Block Chaining (CBC) MAC Schemes.**

*Secret prefix* appended to message prior to iterated hash—can forge any tag if we can obtain the MAC tag for the empty message, by applying the iterated hash to the new message appended to the end.

*Secret suffix method*—since $H$ is an iterated hash function, if we can find a collision for some $x$ and $x'$, then $H(x\|k) = H(x'\|k)$. This presumes that $H$ is not collision-resistant.

## 7. Public-private Key Cryptography.

Drawbacks with SKES: key establishment is difficult.

1. Could distribute the key point-to-point, but this isn't practical.
2. Could use a trusted third party, but then the TTP must be unconditionally trusted, making it an attractive target for attacks, and a critical reliability/potential bottleneck point.

Also, key management can be difficult; in a network on $n$ users, each user must store $n-1$ secret keys.

Last, with a shared key, non-repudiation is impractical, because neither party can be shown to be the source of a message. (Alternatively, requires the involvement of a TTP, with whom each party maintains a shared key.)

**Advantages and Disadvantages of Public-private Key Cryptography.**

On the plus side:

- No requirement for secret channel.
- Each user only has one key pair, simplifying key management.
- Facilitates non-repudiation services (with digital signatures).

On the minus side:

- Public keys tend to be larger than symmetric keys.
- Public-private key schemes tend to be slower than their symmetric counterparts.

The shining example of public-private key cryptography: RSA.

## 8. Digital Signatures.

Different attacks on the signature scheme:

- *Total break:* The adversary recovers Alice's private key, or finds a method to systematically forge Alice's signatures.
- *Selective forgery:* The adversary forges Alice's signature for some selected subset of messages.
- *Existential forgery:* The adversary forges Alice's signature for a single message.

Attack model for the adversary:

- *Key-only attack:* The adversary only knows Alice's public key.
- *Known-message attack:* The adversary knows some valid message/signature pairs.
- *Chosen-message attack:* The adversary has access to a signing oracle, from which it can obtain signatures for some messages of its choosing.

**Definition.** A signature scheme is said to be *secure* if it is existentially unforgeable by a computationally-bounded adversary who launches a chosen-message attack.

**Comparing RSA and DSA Signatures at the 128-bit Security Level.**

Public Key Size:

- RSA: $n = 3072$ bits
- DSA: $p = 3072$ bits, $q = 256$ bits

Signature Size:

- RSA: 3072 bits
- DSA: 512 bits

Signature Verification Speed:

- RSA: Fast; only a few modular multiplications (especially if $e = 3$)
- DSA: Slower; two modular exponentiations

Signature Generation Speed:

- RSA: Slow; requires full exponentiation
- DSA: Requires some time to generate $k$ and $r$, but this can be done in advance; computation of $s$ is very fast

Security:

- RSA: Based on the difficulty of integer factorization
- DSA: Based on the difficulty of the discrete log problem

## 9. Electronic Cash.

It's cash for the digital age!

Presuming we could develop such a thing (hello Royal Mint of Canada?), we would want it to have similar desirable characterics to regular paper (plastic) cash:

- *Recognizable* as legal tender
- *Portable*, or easily carried
- *Transferable*, without the involvement of a financial network
- *Divisible*, e.g. having the ability to make change
- *Unforgeable*, so duplication is impossible (for payee and Bank)
- *Untraceable*, so it is hard to keep track of where money is spent (for payer)
- *Anonymous*, with no record of who spent the money (for payer)
- No double spending (for payee and Bank)

We theorize two kinds of payments: *on-line payments*, where the Bank must be contacted to verify the validity of the payment prior to the completion of the transaction, and *off-line payments*, where the payment is submitted for verification and deposit after the transaction is completed. The off-line system is almost certainly preferable to the on-line one.

## 10. Discrete Logarithm Cryptography.

**The Discrete Log Problem.**

Let $p$ be a prime, and let $g$ be a generator of $Z_p{}^*$, and let $h \in Z_p{}^*$. Given $p, g$, and $h$, find $a \in [0, p-2]$ such that $g^a = h \pmod p$. In this case, we write $a = \log_g h$.

**Discrete Log Cryptosystems.**

Let $p$ be an odd prime, and let $q$ be a prime divisor of $p-1$. Let $g$ be an element in $\mathbb{Z}_p{}^*$ of order $q$.

We now have a variant on the discrete log problem: given $p, q, g$, and $h \in \langle g \rangle$ (where $\langle g \rangle$ denotes the subgroup generated by $g$), find the integer $a \in [0, q-1]$ such that $h = g^a \pmod p$.

**Attacks on the DLP.**

1. Exhaustive search.
2. Shanks' Baby-step Giant-step Algorithm.
   Let $m = \lceil \sqrt{q} \rceil$, and write $i = am + b$, where $a, b \in [0, m-1]$.
   Then $g^i = g^{am+b} = g^{am} g^b \pmod p$, and thus $hg^{-b} = g^{am} \pmod p$.
   (i) Compute $m$.
   (ii) For each $b \in [0, m-1]$, compute $c = hg^{-b} \pmod p$, and store the entry $(c, b)$ in a table.
   (iii) For each $a \in [0, m-1]$, compute $g^{am} \pmod p$, and look it up in the table until a match is found.

   Then if $g^{am} = hg^{-b} \pmod p$, $\log_g h = am + b$.

   Runtime: worst case has $O(\sqrt{q})$ modular multiplications
   Space: Up to $O(\sqrt{q})$ table rows
3. Pollard's Algorithm: $O(\sqrt{q})$ modular multiplications, negligable space requitements.
4. Number Field Sieve.

The third and fourth algorithms are the fastest solutions known. Thus, in order to maintain 80-bit security, we must pick $q \approx 2^{160}$, $p \approx 2^{1024}$.

## 11. Elliptic Curve Cryptography.

**Definition.** An *elliptic curve* $E$ over a field $\mathbb{F}$ is defined by an equation of the form $E\colon Y^2 = X^3 + aX + b$, where $a, b \in \mathbb{F}$, $4a^3 + 27b^2 \neq 0$.

The *set of points* on $E$ is $E(\mathbb{F}) = \{(x, y) : x, y \in \mathbb{F},\ x^3 + ax + b = y^2\} \cup \infty$.

We can perform operations on particular points on an elliptic curve. For instance, we define addition and scalar multiplication as follows:

For $P \neq Q$, $(x, y) \neq (x, -y)$—

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

$$(x_3, y_3) = (x_1, y_1) + (x_2, y_2), \text{ where}$$
$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = -y_1 + \lambda(x_1 - x_3)$$

For $P = Q$, then $P + Q = 2P$—

$$\lambda = \frac{3{x_1}^2 + a}{2y_1}$$

$$(x_3, y_3) = (x_1, y_1) + (x_2, y_2), \text{ where}$$
$$x_3 = \lambda^2 - 2x_1$$
$$y_3 = -y_1 + \lambda(x_1 - x_3)$$

For $P = (x, y) = (x, -y) = Q$—

$$P + Q = \infty$$

We see that with addition defined in this manner, $(E(\mathbb{F}), +)$ forms an Abelian group, because the following properties are satisfied:

1. $P + \infty = \infty + P = P, \ \forall \ P \in E(\mathbb{F})$ (identity)
2. $P + (-P) = \infty, \ \forall \ P \in E(\mathbb{F})$ (inverse)
3. $P + Q = Q + P, \ \forall \ P, Q \in E(\mathbb{F})$ (commutativity)
4. $(P + Q) + R = P + (Q + R), \ \forall \ P, Q, R \in E(\mathbb{F})$ (associativity)

In fact, we can show that for any $P \in E(\mathbb{Z}_p), P \neq \infty$, $P$ is a generator for the group. That is, the group is cyclic.

**Elliptic Curve Discrete Log Problem.**

Given an elliptic curve $E$ defined over a finite field $\mathbb{Z}_p$, the prime $p$, and points $P$ and $Q$, with $|E(\mathbb{Z}_p)| = q$, find $i \in [0, q-1]$ such that $iP = Q$.

The fastest solution known to this problem is Pollard's algorithm; there is no known number field sieve attack for the ECDLP. So ECC is secure at smaller computational sizes compared to other forms of cryptography.

**Elliptic Curve Diffie-Hellman Exchange.**

We choose public parameters $E, p, q, P \in E(\mathbb{Z}_p)$.

Then Alice chooses $a \in [1, q-1]$, and computes $A = aP$, and Bob chooses $b \in [1, q-1]$, computing $B = bP$. Both compute $K = aB = bA = abP$, and this is the shared secret. Then both Alice and Bob delete their private keys.

One nice feature of ECDHE is that *forward secrecy* is provided, because the private keys cannot be recovered in the future in order to break the ciphertext. SSL in general practice, for instance, does not provide this, because the private keys can be obtained.

## 12. Pseudo-Random Bit Generation.

We touched on PRBGs earlier when we discussed the RC4 encryption protocol.

**Definition.** A *random bit generator* is a device that outputs a sequence of independent, unbiased bits.

To be secure, the random bits must be *unpredictable* to an active adversary.

Often, true random bits can be obtained by sampling many random sources (e.g. a microphone, I/O samples, etc.) and then hashing the concatenation of the sources.

**Definition.** A *pseudo-random bit generator* $A$ is a deterministic algorithm that, on input of a (truly) random seed $s$ of length $k$ bits, outputs a bit string $A(s)$ of length $m(k) \gg k$ bits in length which appears random.

**Definition.** A PRBG is said to be *cryptographically strong* if it passes all polytime statistical tests. That is, there is no polytime algorithm which can correctly distinguish (with probability significantly greater than 50%) output sequences of the PRBG from truly random sequences.

**Definition.** A PRBG is said to *pass the next-bit test* if there is no polytime algorithm which, on input of the first $\ell$ bits of the output sequences of the PRBG, can correctly predict the $\ell+1$st bit with probability significantly greater than 50%.

**Yao's Theorem.** A PRGB is cryptographically strong if and only if it passes the next-bit test.

**Generic Construction of a PRGB.**

Let $D$ be a finite set, $f\colon D \to D$ be an efficiently computable permutation, and $B\colon D \to \{0,1\}$ be a Boolean predicate, where it is computationally infeasible to correctly compute $B(x)$ given $x \in_R D$, but it is easy to compute $B(x)$ given $y = f^{-1}(x)$.

To see what we mean by this, define an example $B$ over $D = \mathbb{Z}_n$. Let $(n, e)$ be a public RSA key, and let $f$ represent exponentiation by $e$. Let

$$B(x) = \begin{cases} 1 \text{ if } x^d \pmod n \text{ is even} \\ 0 \text{ if } x^d \pmod n \text{ is odd} \end{cases}$$

Clearly, $x$ is easy to compute given $x$ and $f^{-1}(x)$ (since $f^{-1}(x) = x^d \pmod n$), but $B(x)$ is hard to compute given only $x$.

We can construct the generic PRBG $G$ as follows:

1. Select $x_0 \in_R D$ (the seed).
2. Compute $x_1, x_2, \ldots, x_m$, where $x_{i+1} = f(x_i)$.
3. Apply $B$:

$$\begin{aligned} b_{m-1} &= B(x_1) \\ b_{m-2} &= B(x_2) \\ &\vdots \\ b_0 &= B(x_m) \end{aligned}$$

4. The output sequence is $b_0, b_1, \ldots, b_{m-1}$.

**Theorem.** Such a $G$ is a cryptographically strong PRBG.