Introduction (Mihir Bellare Text/Notes: http://cseweb.ucsd.edu/users/mihir/cse207/)

Cryptography provides:
- Data Privacy
- Data Integrity and Authenticity

Crypto-systems all around us
- ATM machines
- Remote logins using SSH
- Web browsers (https invokes Secure Socket Layer (SSL))

Cryptography ensures security of communication across an insecure medium

Adversary is clever person with a powerful computer
Introduction

Ideal channel doesn’t exist

Cryptonium pipe: Cannot see inside or alter content.

Impenetrable pipe between sender and receiver that no one else can see inside or change what’s there

Cryptography cannot achieve all the properties of an ideal pipe
Instead, a few central security goals are targeted, privacy and authenticity/integrity

Protocols are defined that are designed to achieve these security goals
Introduction

A protocol is a collection of programs, one for the sender and one for the receiver. Sender program uses a cryptographic key to encapsulate while receiver program reverses the process.

A trust model specifies who has what keys:
- Symmetric (shared-key) trust model
- Asymmetric (public-key) trust model

In the symmetric model, the sender and receiver share a secret key that the adversary does not know.

Note the secure distribution of the key is not part of the symmetric model (or any model), rather the model specifies how the key is generated and used.

Distributing and keeping a key secure is the domain of computer systems security.

A protocol that is used to provide privacy in the symmetric setting is called a symmetric encryption scheme.
Symmetric Encryption Scheme

For a scheme \( \Pi \), we must specify three algorithms

\[ \Pi = (H, E, D) \]

where \( E \) represents the encryption algorithm
\( H \) represents the algorithm that generates the key \( K \)
\( D \) represents the decryption algorithm

\( M \) represent the message -- also called the plaintext

Sender encrypts the \( M \) by applying algorithm \( E \) to \( K \) and \( M \) to obtain ciphertext

Decryption may fail (if sent by adversary) and produce the upside-down \( \Pi \)

where \( K_e = K_d \)
Symmetric Encryption Scheme

Note that no security scheme can guarantee privacy
It can only be evaluated on the grounds that it provides some probability of preventing the adversary from breaking it

The message authentication problem, in which the receiver can verify the sender, is addressed in the symmetric scheme using a message authentication code (MAC)

\[ \Pi = (H, T, V) \quad \text{(Three algorithms)} \]

Sender computes a ’tag’ or MAC, \( \sigma \), by applying \( T \) using the shared key \( K \) and message \( M \) and then transmits the pair \((M, \sigma)\)

The receiver uses the key \( K \) to check if the tag is OK by applying the verification algorithm, \( V \) with \( K, M \) and \( \sigma \).

If algorithm returns ’1’, message is accepted as authentic
Symmetric Encryption Scheme

Figure 1.3: Symmetric encryption. The sender and the receiver share a secret key, $K$. The adversary lacks this key. The message $M$ is the plaintext; the message $C$ is the ciphertext.

Figure 1.4: A message authentication code. The tag $\sigma$ accompanies the message $M$. The receiver $R$ uses it to decide if the message really did originate with the sender $S$ with whom he shares the key $K$. 
Asymmetric Encryption Scheme

In the asymmetric setting, an individual possesses a pair of keys-- a public key, \( pk \), and an associated secret key, \( sk \)

Also called **public-key** setting

The public key is made *publicly known*, e.g., placed in phone book, and bound to its identity

For encryption, the sender is **assumed** to be able to obtain an *authentic copy* \( pk_R \) of the receiver’s public key (adversary also knows \( pk_R \))

Sender computes ciphertext \( C \leftarrow E_{pkR}(M) \) and sends \( C \) to receiver

Receiver computes \( M \leftarrow D_{skR}(C) \) using the receiver’s secret key \( sk_R \)

This is a very useful mechanism

This allows you to look up the receiver’s public key and send him/her a message that no one else can read -- even if you’ve never met the receiver!
Asymmetric Encryption Scheme

The tool used for solving the message-authentication problem in the asymmetric setting is a digital signature.

The sender has a public key $pk_S$ and a corresponding secret key $sk_S$.

The receiver (and adversary) is assumed to know the key $pk_S$ and that it belongs to $S$. 

Figure 1.5: Asymmetric encryption. The receiver $R$ has a public key, $pk_R$, which the sender knows belongs to $R$. The receiver also has a corresponding secret key, $sk_R$. 

The sender has a public key $pk_S$ and a corresponding secret key $sk_S$. 

The receiver (and adversary) is assumed to know the key $pk_S$ and that it belongs to $S$. 

The tool used for solving the message-authentication problem in the asymmetric setting is a digital signature.
Asymmetric Encryption Scheme

The sender attaches to the message $M$ some extra bits $\sigma$ (called the signature).

The signature is computed as a function of $M$ and $sk_S$ using a signing algorithm (Sign).

The receiver on receipt of $M$ and $\sigma$, checks that it is OK using the sender’s public key $pk_S$ by applying a verification algorithm $V - V$ either accepts or rejects.

$$\Pi = (K, \text{Sign, } V) \quad \text{(Three algorithms)}$$

![A digital signature scheme. The signature $\sigma$ accompanies the message $M$. The receiver $R$ uses it to decide if the message really did originate with the sender $S$ with his public key $pk_S$.](image-url)
Asymmetric Encryption Scheme

In summary:

<table>
<thead>
<tr>
<th>message privacy</th>
<th>symmetric (a.k.a. private-key) encryption</th>
<th>asymmetric (a.k.a. public-key) encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>message authenticity</td>
<td>message authentication code (MAC)</td>
<td>digital signature scheme</td>
</tr>
</tbody>
</table>

Figure 1.7: Summary of main goals and trust models.

One difference between MAC and the digital signature concerns the notion of non-repudiation

With MAC, anyone who can verify a tagged message can also produce one -- which suggests that authenticity can NOT be proved in a court of law.

In contrast, a digitally-signed message, the ONLY person who should be able to produce M that verifies under the public key $pk_S$ is the $S$ herself.

So $S$ cannot claim that the receiver presenting the evidence concocted it.
Asymmetric Encryption Scheme
If signature $\sigma$ authenticates $M$ with $pk_S$, then it is only $S$ that should have been able to
construct $\sigma$
$S$ cannot refute that fact

All that sender $S$ can claim is that key $sk_S$ was stolen -- but that may still leave sender
$S$ responsible

Other Goals of Cryptography
• Pseudorandom Number Generation
  Lots of applications require random numbers, including simulation, efficient algorithms and cryptography (key generation and randomized encryption algo)

  A pseudorandom number generator is deterministic in that it takes a seed and produces a sequence of random numbers, that is repeatable for the same seed

  The seed is a key element -- the task of random number generation is reduced to the task of generating a short random seed
Other Goals of Cryptography

- **Pseudorandom Number Generation**
  The generation of the seed can be done using a Geiger counter or by computing some function of various system parameters such as time and system load.

  The most important element of seed generation is that the process be *completely unpredictable*.

- **Authenticated Key Exchange**
  It is common for an individual to establish a **secure session**.
  For example, remote login to a computer or in a web-browsing session.

  For situations in which a secret key is shared (symmetric) or public keys are available (asymmetric), a secure session can be established using cryptography.

  However, it is not how it is done, rather the parties use their existing keys (*long-lived keys*) to derive a **session key**.

  Which is done through a **authenticated key exchange** protocol.
Other Goals of Cryptography

- Authenticated Key Exchange
  The protocol is based on the exchange of messages designed to provide a *fresh* and *authenticated* key to implement a symmetric cryptography session

- Coin Flipping
  Alice and Bob need to decide who gets the car in a divorce -- Alice proposes that she flip a coin and then have Bob ’call it’

  The challenge is how do you implement a system that assures that neither tries to cheat and that each learns the true outcome of the coin toss

  Solution: Alice puts a random bit $\alpha$ inside an envelope and sends it to Bob -- Bob announces a random bit $\beta$ -- Bob opens the envelope and both compute $\alpha\ xor\ \beta$ (the shared bit)

  The scheme in which Alice puts a value in an envelope and Bob can’t see it is called a *commitment scheme* in cryptography
Other Goals of Cryptography

• Coin Flipping

A simple scheme to implement the *envelope*: Alice puts a "0" in an envelope by choosing two random 500-bit primes $p$ and $q$ subject to

\[
p < q \\
p = 1 \pmod{4} \\
q = 3 \pmod{4}
\]

The product $N = p \times q$ is the commitment to zero, i.e., what Alice would send to commit to "0"
Other Goals of Cryptography

- Coin Flipping

Alice puts a "1" in an envelope by choosing two random 500-bit primes $p$ and $q$ subject to

\[ p < q \]

\[ p = 3 \pmod{4} \]

\[ q = 1 \pmod{4} \]

The product $N = p*q$ is the commitment to "1"

Poor Bob, seeing $N$, would like to determine if the smaller of its two prime factors is congruent to 1 or to 3 (mod 4)

Unfortunately, there is no way of determining that short of factoring $N$

And we don’t know how to factor a 1000 digit number which is the product of 500-digit primes

When Alice wants to decommit $N$ (open the envelope), she announces $p$ and $q$

Bob confirms they are prime (easy), multiply to $N$, and determines the result
What is Cryptography About?

Cryptography is about constructing and analyzing protocols, which are designed to be extremely difficult for adversaries to break

Cryptography problems involve parties (humans, computers, gadgets, etc) playing the role as the good guys

We design protocols for the parties to use

The protocol tells each party how to behave and is typically a (distributed) program

Protocols tell the parties what to do, but not the adversary -- they can do anything

A protocol can be probabilistic (can make random choices) and have state

\[ i \leftarrow \{0, 1, \ldots, n-1\} \]

The former is formalized as ’a random value \( i \) from the set on the right is returned (with all values equally likely) for a value \( n \geq 2 \) selected by a party’
What is Cryptography About?

How do we devise and analyze protocols?

The first step is to understand the **threats** and **goals** of our particular problem.

Once understood, we then attempt to find a protocol solution.

The **adversary** is the agent that embodies the *source* of the threat -- their goal is to defeat our protocol’s goals.

So it is a game -- who is more clever, the protocol designer or the adversary.

In formal analysis, the parties may *vanish* (absorbed in the formalization) but the adversary NEVER vanishes and remains at center stage.

Cryptography is largely about thinking about the adversary -- what can she do? what is she trying to accomplish?

These questions need answers before we can make progress.
Cryptography and Computer Security

Good protocols are the key element of making secure computing systems
Good protocol design is also **hard** and easy to underestimate

Secure systems involve a combination of different factors and define **system security**
An adversary should **not** be able to exploit bugs, break into your system and use your account, buy off your system administrator, steal backup tapes, etc.

The security of a system is only as strong as its weakest link -- for secure systems, all aspects must be addressed
- How do we secure our machines against intruders?
- How do we administer machines to maintain security?
- How do we design good protocols?
A cryptographic protocol is just one piece of the puzzle and is the part we focus on

There are rules
- We can only overcome the adversary by means of protocols (can’t poison her coffee)
- Only *keys* are secret -- protocols should be made *public*
Modern Cryptography

The *systematic* approach to cryptography (where proofs and definitions play a visible role) began with the work of Claude Shannon (father of information theory).

He developed the notion of **perfect security**

Unfortunately, perfect security is not practical because it requires the number of bits in the *key* to be at least as large as the number in the *message*.

Modern cryptography introduces a new dimension: the amount of *computing power* available to the adversary.

Focus on schemes that are breakable *in principle* but NOT in practice.

This takes cryptography from the realm of information theory to computer science.

In particular, to *complexity theory* where we study how hard problems are to solve as a function of computational resources invested.

The following (probabilistic) principle is the basis:

Assuming the adversary uses no more than $t$ computing cycles, her probability of breaking the scheme is at most $t/2^{200}$.
Modern Cryptography

There is no notion of how the adversary operates, what algorithms or techniques are employed -- which is good because we don’t know these.

To understand how we get protocols with such properties, let’s first look at atomic primitives.

Higher level protocols are built on top of atomic primitive

  Protocols address a cryptographic problem by specifying how we encrypt, how we distribute a key.

  Atomic primitives, on the other hand, are generally simpler protocols that have some sort of hardness or security properties, but by themselves are not useful.

Atomic primitives are drawn from two sources:

  • Engineered constructs, e.g., blockciphers such as DES algorithm
  • Mathematical problems, e.g., RSA
Modern Cryptography

Based on the above, modern cryptography is based on computationally hard problems

Originally, \textit{NP-complete} problems were looked at for cryptography but failed because they are hard to solve in the worst case, and easy on average.

A more suitable example is a \textbf{one-way function} \(f: D \rightarrow R\) mapping some domain \(D\) to range \(R\), with the following properties:

- \(f\) is easy to compute, there is an efficient algorithm given \(x\) in \(D\) outputs \(y = f(x)\) in \(R\)
- \(f\) is hard to invert: an adversary \(I\) given a random \(y\) in \(R\) has a hard time figuring out a point \(x\) such that \(f(x) = y\) (as long as her computing time is limited)

As noted above, the latter property will be expressed in terms of \textit{probability}

One-way functions are ubiquitous in the real world, e.g., smashing a mirror is easy.

Number theory provides a very simple one that is based on \textit{multiplication}

The function \(f\) takes as input 2 numbers \(a\) and \(b\), and multiplies them to produce

\[ N = a \times b \]
Modern Cryptography

Interestingly, there is no known algorithm that given a random $N = a*b$, always and quickly recovers a pair of numbers (excluding 1 or $N$) that are factors of $N$

The backwards direction is called the factoring problem -- a problem that has remained unsolved for hundreds of years

A second example: Let $p$ be prime and consider the function $f(x) = g^x \mod p$

This is called the discrete exponentiation function and its inverse the discrete logarithm function: $\log_g(y)$ is the value $x$ such that $y = g^x$

It turns out that there is no known fast algorithm that computes discrete logarithms. Given a large enough $p$, e.g., 1000 bits, the task is infeasible given the state of computing power today

Note, like P vs NP, there is no proof that these are hard functions to invert

We assume they are, but if someone comes up with a fast algorithm, a lot of protocols will fail!