Lecture 1: Introduction and Overview

TTM4135

Relates to Stallings Chapter 1

Spring Semester, 2025

Many thanks to Colin Boyd for the slides

Motivation

What is this course about?

- How does this course run?
- Why is cryptography and network security important?
- What is the connection between cryptography and general information security?
- What is in this course?

Outline

Introduction to the course

Who Needs Cryptography and Network Security?

Role of Cryptography in Information Security

Course outline

Outline

Introduction to the course

Who Needs Cryptography and Network Security?

Role of Cryptography in Information Security

Course outline

Administration

- Responsible professor: Assoc. Prof. Anamaria Costache
- Scientific assistants:
  - PhD student Lea Nürnberger
- Materials for lectures, exercises, assessment items are on Blackboard.

Textbooks and lecture notes

- Recommended textbook: Cryptography and Network Security, William Stallings, 8th Edition.
- Textbook will be useful to back up lectures. It is a little out of date.
- The syllabus for the examination is defined by the lecture slides, not by the textbook.
- Exercises will be useful for exam preparation.
- Many useful resources online - some will be mentioned on Blackboard.

Assessment

Three items:

- ongoing work during semester (20%)
  - weekly online quizzes (10%)
  - practical cryptanalysis exercise (10%)
- lab milestones and report (20%)
- written examination (60%)

Check timetable on Blackboard for submission dates and other details. Note that the timetable may sometimes be updated. It is *your responsibility* to be up to date with all the changes and check Blackboard periodically.

Timetable

- Lecture times
  - Mondays 12:15 – 14:00
  - Friday 8:15 – 10:00
- Additional class time
  - Friday 10:15 – 11:00 will be used for different purposes — *not* a lecture
- Lab
  - Three weeks starting from when the lecture phase has finished

Check timetable on Blackboard. Note that the timetable may sometimes be updated.

Comparison with last year

- All physical lectures, no recordings
- Most topics and overall format unchanged
- Small updates to lecture material
- Piazza is used for online Q&A
- Please give feedback and *volunteer for the reference group*

Outline

Introduction to the course

Who Needs Cryptography and Network Security?

Role of Cryptography in Information Security

Course outline

Outline

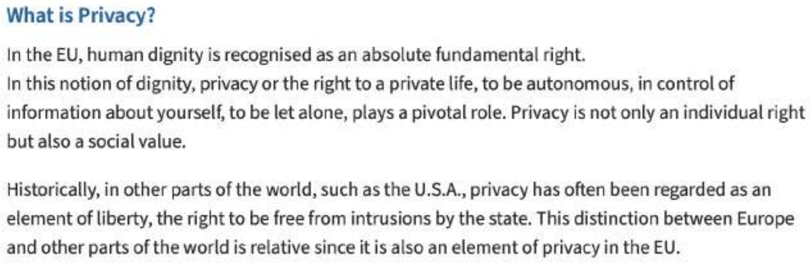
Introduction to the course

Who Needs Cryptography and Network Security?

Role of Cryptography in Information Security

Course outline

What is Privacy?



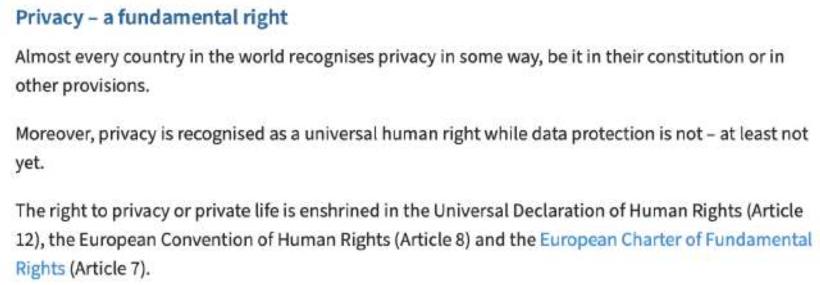
What is Privacy?

In the EU, human dignity is recognised as an absolute fundamental right.
In this notion of dignity, privacy or the right to a private life, to be autonomous, in control of information about yourself, to be let alone, plays a pivotal role. Privacy is not only an individual right but also a social value.

Historically, in other parts of the world, such as the U.S.A., privacy has often been regarded as an element of liberty, the right to be free from intrusions by the state. This distinction between Europe and other parts of the world is relative since it is also an element of privacy in the EU.

https://edps.europa.eu/data-protection/data-protection_en

Privacy – a fundamental right



Privacy – a fundamental right

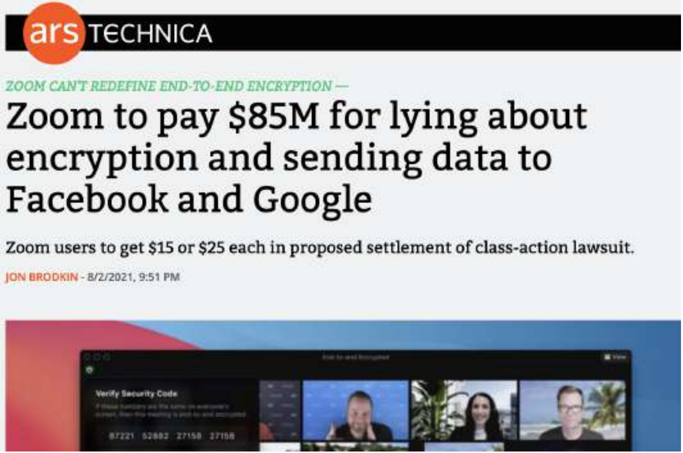
Almost every country in the world recognises privacy in some way, be it in their constitution or in other provisions.

Moreover, privacy is recognised as a universal human right while data protection is not – at least not yet.

The right to privacy or private life is enshrined in the Universal Declaration of Human Rights (Article 12), the European Convention of Human Rights (Article 8) and the European Charter of Fundamental Rights (Article 7).

https://edps.europa.eu/data-protection/data-protection_en

A few recent headlines – February 2021



ars TECHNICA

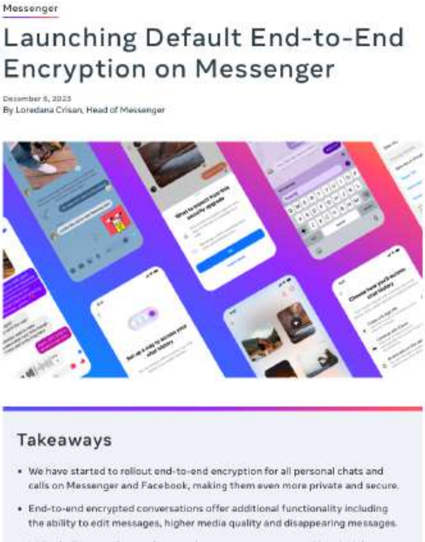
ZOOM CAN'T REDEFINE END-TO-END ENCRYPTION —

Zoom to pay $85M for lying about encryption and sending data to Facebook and Google

Zoom users to get $15 or $25 each in proposed settlement of class-action lawsuit.

JON BRODKIN - 8/2/2021, 9:51 PM

A few recent headlines – December 2023



Messenger

Launching Default End-to-End Encryption on Messenger

December 6, 2023
By Loredana Crisan, Head of Messenger

Takeaways

- We have started to rollout end-to-end encryption for all personal chats and calls on Messenger and Facebook, making them even more private and secure.
- End-to-end encrypted conversations offer additional functionality including the ability to edit messages, higher media quality and disappearing messages.

Outline

Introduction to the course

Who Needs Cryptography and Network Security?

Role of Cryptography in Information Security

Course outline

Outline

Introduction to the course

Who Needs Cryptography and Network Security?

Role of Cryptography in Information Security

Course outline

Defining information security

ISO security architecture definition

"The term *security* is used in the sense of minimizing the vulnerabilities of assets and resources. An asset is anything of value. A *vulnerability* is any weakness that could be exploited to violate a system or the information it contains. A *threat* is a potential violation of security."

- *Information security* can be defined as security where the assets and resources are information systems. This can include data, software and hardware, people and even buildings.

The CIA triad

Traditional definitions of information security are based on three information security goals:

Confidentiality: preventing unauthorised disclosure of information

Integrity: preventing unauthorised (accidental or deliberate) modification or destruction of information

Availability: ensuring resources are accessible when required by an authorised user

OSI Security Architecture X.800

The OSI (Open Systems Interconnection) Security Architecture defines a *systematic approach* to providing security at each layer. It defines security services and security mechanisms that can be used at each of the seven layers of the OSI model to provide security for data transmitted over a network. These services and mechanisms help achieve the CIA goals. Freely downloadable:

`http://www.itu.int/rec/T-REC-X.800-199103-I/e`

- A bit dated now but still worth looking at. Most definitions and terminology still apply.
- Defines *security threats* (or attacks), *security services* and *security mechanisms* and how they are related.

Useful supplement is Internet Security Glossary, RFC 4949.

Passive Threats

Passive threats do not alter information in the system. Such threats may be hard to detect.

Eavesdropping The attacker monitors the communication, for example by sniffing packets or tapping a telephone wire.

Traffic analysis The attacker monitors the amount, source and destination of communication.

Active threats

Active threats alter information in the system. Such threats may be hard to detect.

Masquerade: the attacker claims to be a different entity.

Replay: the attacker sends a message which has already been sent.

Modification of messages: the attacker changes messages during transmission.

Denial of service: the attacker prevents legitimate users from accessing resources

Security services and mechanisms

Security service: a processing or communication service to give a specific kind of protection to system resources

Security mechanism: a method of implementing one or more security services

In this course we look closely at *cryptographic* security mechanisms

Main security services

- *Peer entity authentication* provides confirmation of the claimed identity of an entity.
- *Data origin authentication* provides confirmation of the claimed source (origin) of a data unit (message).
- *Access control* provides protection against unauthorized use of resources.

  Access control service is usually provided in combination with authentication and authorisation services.
- *Data confidentiality* protects data against unauthorised disclosure.
- *Traffic flow confidentiality* protects disclosure of data which can be derived from knowledge of traffic flows.

Main security services (continued)

- *Data integrity* detects any modification, insertion, deletion or replay of data in a message or a stream of messages.
- *Non-repudiation* protects against any attempt by the creator of a message to falsely deny creating the data or its contents.
  X.800 talks about *nonrepudiation of origin* to protect against denial by the sender of a message, and *nonrepudiation of receipt* to protect against denial by the recipient of a message.
- *Availability service* protects a systems against denial of service.

Main security mechanisms

- *Encipherment* is the transformation of data in order to hide its information content. Later in the course we look at both public-key and symmetric-key encryption.
- *Digital signature mechanisms* are cryptographic algorithms which transform data using a signing key. The essential property is that signed data can only be created with the signing key. We will look at standard signature schemes.
- X.800 describes a variety of *access control mechanisms* including access control lists, passwords, or tokens, which may be used to indicate access rights.
- X.800 describes *data integrity mechanisms* as “corruption detection techniques” which can be used with “sequence information”. We will look at the example of message authentication codes.

Main security mechanisms (continued)

- *Authentication exchange* mechanisms are protocols which exchange information to ensure identity of protocol participants. We will study examples such as TLS later.
- *Traffic padding* is spurious traffic generated to protect against traffic analysis. Traffic padding is typically used in combination with encipherment.
- *Routing control mechanism* is the use of specific secure routes.
- The *notarization mechanism* uses a trusted third party to assure the source or receipt of data. The trusted third party is sometimes called a notary.

Relating security services to mechanisms

Mechanism	Encipherment	Digital signature	Access control	Data Integrity	Auth. exchange	Padding	Routing control	Notarization
Service								
Peer entity authentication	✓	✓			✓			
Data origin authentication	✓	✓						
Access control			✓					
Data Confidentiality	✓						✓	
Traffic Flow Confidentiality	✓					✓	✓	
Data Integrity	✓	✓		✓				
Nonrepudiation		✓		✓				✓
Availability				✓	✓			

From Stallings based on X.800. ✓ indicates the mechanism is relevant to provide the service.

Risk management

A key tool in information security management.

1. Identify threats
2. Classify all threats according to likelihood and severity
3. Apply security controls based on cost benefit analysis

For more details see NIST Special Publication 800-30, Guide for Conducting Risk Assessments, or ISO 27000 standards.

Outline

Introduction to the course

Who Needs Cryptography and Network Security?

Role of Cryptography in Information Security

Course outline

Outline

Introduction to the course

Who Needs Cryptography and Network Security?

Role of Cryptography in Information Security

Course outline

Course focus

- Cryptography as a foundation for information security
- Applications of cryptography in network security
- Prominent internet security protocols

Need some mathematics for cryptography, but emphasise usage rather than proofs.
Note that you need to be able to use the mathematical tools!

Course content

- Historical cryptography
- Modern cryptography: block ciphers, stream ciphers, public key, hash and MAC.
- Some maths, particularly to support public key. Modular arithmetic, number theory, elliptic curves.
- Public key infrastructure
- Secure email and messaging
- Transport Layer Security (TLS) protocol (HTTPS) and how it uses all of the cryptography

How to complete this course successfully?

- Show up and participate to the lectures – active participation is encouraged!
- Show up and participate to the exercise classes – active particiation is encouraged!
- Hand in all assignments in a timely manner
  - If anything prevents you from doing so, let us know *as soon as possible*
- *Practice!!* It is very important that you practice in your own time! You will likely not be familiar with a lot of mathematical tools. They are not overly complicated, but you do need to spend some time practising.

Lecture 2: Number Theory, Groups and Finite Fields

TTM4135

Relates to Stallings Chapters 2 and 5

Spring Semester, 2025

Motivation

- Cryptography makes use of mathematics, computer science and engineering
- Mostly the mathematics is *discrete mathematics* because cryptology deals with finite objects such as alphabets and blocks of characters
- We therefore look at modular arithmetic which only deals with a finite number of values
- Understanding the algebraic structure of finite objects helps to build useful cryptographic properties

Outline

Basic Number Theory
- Primes and Factorisation
- GCD and the Euclidean Algorithm
- Modular arithmetic

Groups

Finite Fields

Boolean Algebra

Outline

Basic Number Theory
- Primes and Factorisation
- GCD and the Euclidean Algorithm
- Modular arithmetic

Groups

Finite Fields

Boolean Algebra

Factorisation

- Let $\mathbb{Z}$ denote the set of integers
- For *a* and *b* in $\mathbb{Z}$, we say that *a* divides *b* (or *a* is a factor of *b*, or write $a|b$) if there exists *k* in $\mathbb{Z}$ such that $ak = b$
- An integer $p > 1$ is said to be a *prime number* (or simply a *prime*) if its only positive divisors are 1 and *p*
- We can test for prime numbers by trial division (up to the square root of the number being tested)
- In a later lecture we will look at a more efficient way to check for primality

Basic Properties of Factors

1. If a divides b and a divides c, then a divides $b + c$
2. If p is a prime and p divides ab, then p divides a or b

Euclidean division

For a and b in $\mathbb{Z}$, $a > b$, there exist unique q and r in $\mathbb{Z}$ such that:

$$a = bq + r$$

where $0 \leq r <\mid b \mid$.

Outline

Basic Number Theory

Primes and Factorisation

GCD and the Euclidean Algorithm

Modular arithmetic

Groups

Finite Fields

Boolean Algebra

Greatest common divisor (GCD)

The value d is the GCD of a and b, written $\gcd(a, b) = d$, if all of the following hold:

1. d divides a and b
2. if c divides a and b then c divides d
3. $d > 0$

We say that a and b are *relatively prime*, or *co-prime* if $\gcd(a, b) = 1$

Euclidean algorithm

One can find $d = \gcd(a, b)$.
Let q_i be the quotient and r_i be the remainder in the following.

$$\begin{aligned}
a &= bq_1 + r_1, \text{for } 0 < r_1 < b \\
b &= r_1 q_2 + r_2, \text{for } 0 < r_2 < r_1 \\
r_1 &= r_2 q_3 + r_3, \text{for } 0 < r_3 < r_2 \\
&\vdots \\
r_{k-2} &= r_{k-1} q_k + r_k, \text{for } 0 < r_k < r_{k-1} \\
r_{k-1} &= r_k q_{k+1}, \text{where } r_{k+1} = 0
\end{aligned}$$

Then $d = r_k = \gcd(a, b)$.

```
Data: $a, b$
Result: $\gcd(a, b)$
$r_{-1} \leftarrow a$;
$r_0 \leftarrow b$;
$k \leftarrow 0$;
while $r_k \neq 0$ do
    $q_k \leftarrow \lfloor \frac{r_{k-1}}{r_k} \rfloor$;
    $r_{k+1} \leftarrow r_{k-1} - q_k r_k$;
    $k \leftarrow k + 1$;
end
$k \leftarrow k - 1$;
return $r_k$
```

Algorithm 1: Euclidean algorithm

Back substitution (extended Euclidean algorithm)

- By *back substitution* in the Euclidean algorithm we can find integers x and y where

$$ax + by = d = r_k.$$

- Starting with the penultimate line in the algorithm, $r_{k-2} = r_{k-1}q_k + r_k$, we can compute

$$r_k = r_{k-2} - r_{k-1}q_k.$$

Then we replace r_{k-1} in this equation from the next line up, $r_{k-1} = r_{k-3} - r_{k-2}q_{k-1}$ to get

$$\begin{aligned} r_k &= r_{k-2} - (r_{k-3} - r_{k-2}q_{k-1})q_k \\ &= r_{k-2}(1 + q_{k-1}q_k) - r_{k-3}q_k \end{aligned}$$

- Now we can use this equation to replace r_{k-2} from the line before that, and continue in the same way.
- Finally replacing r_1 by $r_1 = a - bq_1$ from the first line gives us r_k in terms of a multiple of a and a multiple of b.
- We will be particularly interested in the case where $r_k = d = 1$.

Outline

Basic Number Theory

Primes and Factorisation

GCD and the Euclidean Algorithm

Modular arithmetic

Groups

Finite Fields

Boolean Algebra

Modular arithmetic

Definition

b is a residue of a modulo n if $a - b = kn$ for some integer k.

$$a \equiv b \pmod{n} \iff a - b = kn.$$

Given $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then

1. $a + c \equiv b + d \pmod{n}$
2. $ac \equiv bd \pmod{n}$
3. $ka \equiv kb \pmod{n}$

Note

This means we can always reduce the inputs modulo n *before* performing multiplication or addition.

Residue class

Definition

The set $\{r_0, r_1, \ldots, r_{n-1}\}$ is called a *complete set of residues* modulo n if, for every integer a, $a \equiv r_i \pmod{n}$ for exactly one r_i

- The numbers $\{0, 1, \ldots, n-1\}$ form a complete set of residues modulo n since we can write any a as

$$a = qn + r \text{ for } 0 \leq r \leq n-1$$

- We usually choose this set as the complete set of residues and denote it:

$$\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$$

Notation: $a \bmod n$

We write

$$a \bmod n$$

to denote the unique value a' in the complete set of residues $\{0, 1, \ldots, n-1\}$ with

$$a' \equiv a \pmod{n}$$

In other words, $a \bmod n$ is the remainder after dividing a by n

Groups

A *group* is a set, G, with a binary operation, $\cdot$, satisfying the following conditions:

- Closure: $a \cdot b \in G$ for all $a, b \in G$
- Identity: there exists an element, 1, so that $a \cdot 1 = 1 \cdot a = a$ for all $a \in G$
- Inverse: for all $a \in G$ there exists an element, b, so that $a \cdot b = 1$ for all $a \in G$
- Associative: for all $a, b, c \in G$ that $(a \cdot b) \cdot c = a \cdot (b \cdot c)$

We will only be looking at commutative (or abelian) groups which satisfy also:

- Commutative: for all $a, b \in G$ that $a \cdot b = b \cdot a$

Cyclic groups

- The *order of a group G*, often written $|G|$, is the number of elements in G
- We write g^k to denote repeated application of g using the group operation — for example $g^3 = g \cdot g \cdot g$. The *order of an element $g \in G$*, often written $|g|$, is the smallest (non-zero!) integer k with $g^k = 1$
- A group element g is a *generator* for G if $|g| = |G|$
- A group is *cyclic* if it has a generator

Cyclic groups are important in cryptography because if we construct a group G with large order then we can be sure that a generator g can also take on the same large number of values.

Computing inverses modulo n

- The inverse of a, if it exists, is a value x such that

$$ax \equiv 1 \pmod{n}$$

  and is written $a^{-1} \bmod n$.
- In cryptosystems we often need to find inverses so that we can decrypt, or undo, certain operations.

Theorem

Let $0 < a < n$. Then a has an inverse modulo n if and only if $\gcd(a, n) = 1$.

Modular inverses using Euclidean algorithm

- To find the inverse of a we can use the Euclidean algorithm which is very efficient
- Remember that we want to solve for x, given a:

$$ax \equiv 1 \pmod{n}$$

- Since $\gcd(a, n) = 1$ we can find $ax + ny = 1$ for integers x and y by Euclidean algorithm. Therefore:

$$\begin{aligned} ax &= 1 - ny \\ ax &\equiv 1 \pmod{n} \end{aligned}$$

$\mathbb{Z}_p^*$

- A complete set of residues modulo any prime p with the 0 removed forms a group under multiplication denoted $\mathbb{Z}_p^*$.
- Some useful properties:
  - The order of $\mathbb{Z}_p^*$ is $p - 1$
  - $\mathbb{Z}_p^*$ is cyclic
  - $\mathbb{Z}_p^*$ has many generators in general
- $\mathbb{Z}_p^*$ can be represented as the multiplicative group of integers $\{1, 2, \ldots, p - 1\}$

Finding a generator of $\mathbb{Z}_p^*$

- A *generator* of $\mathbb{Z}_p^*$ is an element of order $p-1$
- A general theorem of algebraic groups (Lagrange) implies that the order of any element must exactly divide $p-1$
- To find a generator of $\mathbb{Z}_p^*$ we can choose a value g and test it as follows:
  1. compute all the distinct prime factors of $p-1$ and call them $f_1, f_2, \ldots, f_r$
  2. then g is a generator as long as $g^{(p-1)/f_i} \neq 1 \bmod p$ for $i = 1, 2, , \ldots, r$

Groups for composite modulus: $\mathbb{Z}_n^*$

- For any n, which may or may not be prime, we can define $\mathbb{Z}_n^*$ to be the group of residues which have an inverse under multiplication
- $\mathbb{Z}_n^*$ is a group but is not cyclic in general
- Finding the order of $\mathbb{Z}_n^*$ is difficult in general

Fields

A *field* is a set, F, with two binary operations, $+$ and $\cdot$, satisfying the following conditions:

- F is a commutative group under the + operation, with identity element denoted 0
- $F \setminus \{0\}$ is a commutative group under the $\cdot$ operation
- Distributive: for all $a, b, c \in F$:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

Finite fields

- For secure communications we are usually only interested in fields with a finite number of elements
- A famous theorem says that finite fields exist of size p^n for any prime p and positive integer n, and that no finite field exists of other sizes
- The most interesting cases for us are fields of size p for a prime p and fields of size 2^n for some integer n

Finite field $GF(p)$

- We often write $\mathbb{Z}_p$ instead of $GF(p)$
- Multiplication and addition are done modulo p
- Multiplicative group is exactly $\mathbb{Z}_p^*$
- Later in the course we will see some public key encryption and digital signature schemes using $GF(p)$

Finite field $GF(2)$

- $GF(2)$ is the simplest field. It has only two elements.
- Addition is binary addition modulo 2. This is the same as the logical XOR (exclusive-OR) operation
- Since there is only one non-zero element we have a trivial multiplicative group with the single element 1.
- We often use XOR in cryptography, usually written $\oplus$. For bit strings a and b we write $a \oplus b$ for the bit-wise XOR. For example,

$$101 \oplus 011 = 110$$

Finite field $GF(2^n)$

- Arithmetic in these fields can be considered as polynomial arithmetic where the field elements are polynomials with binary coefficients
- This allow us to equate any n-bit string with a polynomial in a natural way: for example 00101101 $\leftrightarrow x^5 + x^3 + x^2 + 1$
- The field can be represented in different ways by use of a *primitive polynomial* $m(x)$
- Addition and multiplication is defined by polynomial addition and multiplication modulo $m(x)$
- Polynomial division can be done very efficiently in hardware using shift registers

Arithmetic in $GF(2^8)$

- This field is used for calculations in the AES block cipher
- To add two strings we add their coefficients modulo 2 (exclusive or)
- Multiplication is done with respect to a generator polynomial which for AES is chosen as:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

- To multiply two strings we multiply them as polynomials and then take their remainder after dividing by $m(x)$

Boolean values

- A Boolean variable *x* takes the values of 1 or 0 representing *true* or *false*
- A Boolean *function* is any function with range (output) in the set $\{0, 1\}$
- Boolean functions are often represented by a *truth table*
- Each row in the table defines one possible input (tuple) and the associated output value

Boolean operations

- Logical AND: equivalent to multiplication modulo 2

x_1	x_2	$z = x_1 \wedge x_2$
1	1	1
1	0	0
0	1	0
0	0	0

- Logical OR:

x_1	x_2	$z = x_1 \vee x_2$
1	1	1
1	0	1
0	1	1
0	0	0

Negation

Truth table

x	$\neg x$
1	0
0	1

We can also write $\neg x = x \wedge 1$

Lecture 3: Classical Encryption

TTM4135

Relates to Stallings Chapter 3

Spring Semester, 2025

Motivation

Apart from their intrinsic interest we study historical ciphers in order to:

- establish basic notation and terminology;
- introduce basic cryptographic operations that are still used as building blocks for modern cryptographic algorithms;
- explore the typical attacks and adversary capabilities that our cryptosystems should defend against.

Outline

Introduction

- Basic Definitions
- Cryptanalysis
- Statistics of Natural Language

Transposition ciphers

Simple Substitution Ciphers

- Caesar Cipher
- Random Simple Substitution Cipher

Polyalphabetic Substitution

- Vigenère cipher
- Other polyalphabetic ciphers

Outline

Introduction
- Basic Definitions
- Cryptanalysis
- Statistics of Natural Language

Transposition ciphers

Simple Substitution Ciphers
- Caesar Cipher
- Random Simple Substitution Cipher

Polyalphabetic Substitution
- Vigenère cipher
- Other polyalphabetic ciphers

Terminology

The science of *cryptology* is dual-faceted, comprising of:

- *cryptography* - the study of designing cryptosystems, and
- *cryptanalysis* - the study of breaking cryptosystems.

In practice both facets are usually studied together.

Confidentiality and authentication

- Cryptography is the science of *secret writing*. It concerns transformations of data which depend on a secret called the *key*.
- Cryptography can be used to provide *confidentiality* and to provide *authentication* (or *integrity*).
- When used for confidentiality a key is needed in order to *read* the message.
- When used for authentication a key is needed in order to *write* the message.

Cryptosystems

A cryptosystem consists of:

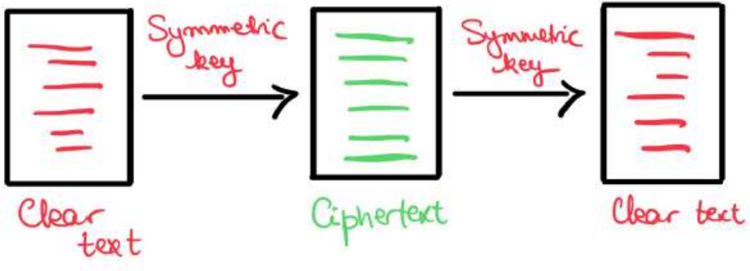
- a set of plaintexts (holding the original message);
- a set of ciphertexts (holding the encrypted message);
- a set of keys;
- a function which transforms plaintext into ciphertext (called *encryption*);
- an inverse function which transforms ciphertext back into plaintext (called *decryption*).

The encrypted message is the ciphertext.

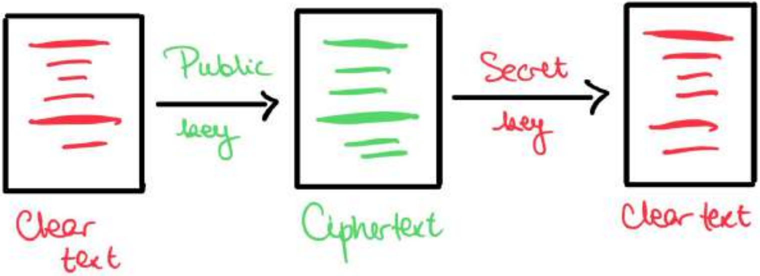
Symmetric and asymmetric cryptography

- Symmetric key cipher (also known as secret key cipher):
  - Encryption and decryption keys known only to the sender and receiver.
  - Requires a secure channel for transmission of the cryptographic key.
- Asymmetric key cipher (also known as public key cipher):
  - Each participant has a public key and a private key.
  - May allow for both encryption of messages and creation of digital signatures.
  - We study public key ciphers in a later lecture.

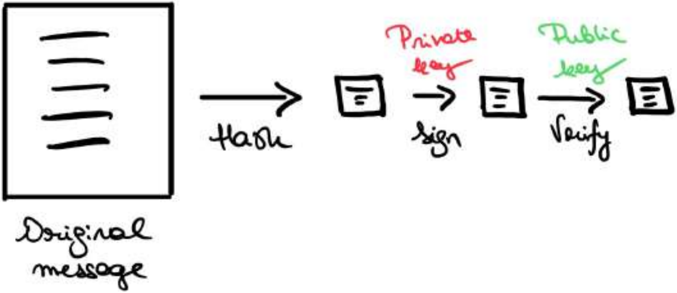
Symmetric Key encryption



Public Key encryption



Digital signatures



Notation for symmetric encryption algorithms

E	=	Encryption function
D	=	Decryption function
M	=	Message or Plaintext
C	=	Cryptogram or Ciphertext
K	=	Shared secret key

Encryption: $C = E(K, M)$

Decryption: $M = D(K, C)$

Outline

Introduction

- Basic Definitions
- Cryptanalysis
- Statistics of Natural Language

Transposition ciphers

Simple Substitution Ciphers

- Caesar Cipher
- Random Simple Substitution Cipher

Polyalphabetic Substitution

- Vigenère cipher
- Other polyalphabetic ciphers

Methods of cryptanalysis

There are many methods available to an adversary who wishes to break a cryptosystem. In general we need to consider the following.

- What resources the adversary has available. This includes the computational capability of the adversary. It may also include access to various inputs and outputs of the system.
- What the adversary is aiming to achieve. This may be to retrieve the whole of the secret key or it may be as little as distinguishing two message (such as *yes* or *no*.)

Exhaustive key search

- The most basic method of attack is *exhaustive key search*, also called brute-force attack, in which the adversary tries all possible keys.
- We cannot prevent this attack so all cryptosystems must have enough keys to make exhaustive search too difficult computationally.
- Note that:
  - it may be possible for the adversary to find the key without trying exhaustive search;
  - the adversary may be able to break the cryptosystem without finding the key.

Prevention of exhaustive key search is a *minimum standard*.

Attack classification

1. **Ciphertext Only attack:** The attacker has available only the intercepted ciphertext.
2. **Known Plaintext attack:** The attacker knows a small amount of plaintext and its ciphertext equivalent.
3. **Chosen Plaintext attack:** The attacker can obtain the ciphertext equivalent of some plaintext which can be selected by the attacker; i.e. the attacker has an “inside encryptor” available.
4. **Chosen Ciphertext attack:** The attacker can obtain the plaintext equivalent of some ciphertext which can be selected by the attacker; i.e. the attacker has an “inside decryptor” available.

Which attacks should be prevented?

- A cryptosystem which can be practically attacked using only ciphertexts, is generally considered to be highly insecure.
- The modern standard is that a cryptosystem should be secure against chosen plaintext and chosen ciphertext attacks.
- History shows that chosen ciphertext attacks can often be practical to set up for an attacker.

Kerckhoffs' Principle

This principle says that an attacker has complete knowledge of the how the cryptosystem works. The decryption key is the only unknown to the attacker.

- History has shown that Kerckhoff's Principle is a reasonable assumption.
- Using a secret, non-standard algorithm can cause severe problems. This would be an example of *security through obscurity*.

Outline

- Introduction
  - Basic Definitions
  - Cryptanalysis
  - Statistics of Natural Language
- Transposition ciphers
- Simple Substitution Ciphers
  - Caesar Cipher
  - Random Simple Substitution Cipher
- Polyalphabetic Substitution
  - Vigenère cipher
  - Other polyalphabetic ciphers

Alphabets

- In our historical ciphers, we need to define the *alphabet* for the plaintext and ciphertext (they are usually the same).
- We will use the Roman alphabet: A, B, C, . . . , Z. Sometimes we include the space, sometimes we use both upper and lower case, sometimes we include punctuation.
- For some ciphers we map the alphabet to numbers and usually assume: $A = 0, B = 1, C = 2, \ldots, Z = 25$. If the space is included we map it to the number 26.
- Note that a real-world attacker needs to work out the alphabet.

Statistical attacks

- To a large extent the statistical attacks depend on using the redundancy of the plaintext. Can you read this?

  `TDY S VRY CLD`

- In written text considerable information is available from the distribution of single letters, digrams (double letters) and trigrams (triple letters) to help in the attack.
- The exact statistics of a language will vary according to what sample is taken.

Sample statistics for English

- The following statistics give a typical distribution of English text. This particular distribution was calculated on a text passage of 143000 characters.
- In order to simplify the statistics, the text is restricted to a plaintext alphabet of 27 characters: {ABCDEFGHIJKLMNOPQRSTUVWXYZ∇}. Here ∇ represents the space character.
- The proportions shown are relative; for example the ∇ character accounts for 14.6% of all characters while 2.3% of all digrams are the E∇ digram.

Frequency (percentage) of characters and digrams

∇	14.6	A	7.0	H	2.6	V	1.3	Z	0.1
E	10.1	R	5.2	M	2.5	B	1.3	J	0.1
N	7.8	S	5.1	P	2.5	Y	0.8	Q	0.1
T	7.5	L	3.7	U	2.4	W	0.6		
I	7.1	C	3.5	G	1.7	K	0.2		
O	7.0	D	3.5	F	1.6	X	0.1		

E∇	2.3	D∇	1.7	ES	1.3	RE	1.1
∇A	2.1	TI	1.7	AT	1.3	IO	1.1
ON	1.9	AN	1.6	ND	1.3	∇I	1.1
IN	1.9	EN	1.6	N∇	1.3	ME	1.0
∇T	1.8	TH	1.6	AL	1.2	ER	0.9
S∇	1.7	NT	1.4	HE	1.2	∇O	0.9

These are typical figures but will vary with the source

Basic cipher operations

Most historical ciphers are based on a combination of two basic operations.

Transposition: the characters in the plaintext are mixed up with each other (permuted).

Substitution: each character (or set of characters) is replaced by a different character (or set of characters).

Transposition ciphers

- A transposition cipher permutes characters usually in a fixed period d and permutation f.
- We can consider the plaintext as a matrix of rows of length d
- Generally transposition ciphers can permute rows or columns and output in row or column order.

Simple transposition cipher

- The key is the pair d and f.
- Each block of d characters is re-ordered using the permutation f.
- There are $d!$ permutations of length d. (Remember that $d! = d \times (d-1) \times (d-2) \times \cdots \times 2 \times 1$.)
- When $d = 10$ there are thus $3,628,800$ keys.

Cryptanalysing a transposition cipher

- The frequency distribution of the ciphertext characters is the same as for the plaintext characters. This helps to identify a transposition cipher.
- If the period d is small then transposition ciphers can be solved by hand using the process of anagramming (restoring disarranged characters to their original position).
- We can guess the value of d and write the ciphertext in columns so that there are d columns.
- Knowledge of the plaintext language digrams and trigrams can then optimise trials.
- This process can be automated.

Simple substitution ciphers

- Each character in the plaintext alphabet is replaced by a character in the ciphertext alphabet as defined by a substitution table.
- Simple substitution ciphers are also called *monoalphabetic* substitution ciphers.
- Note that transposition ciphers permute plaintext characters while substitution ciphers permute alphabet characters.
- There are many special cases of simple substitution ciphers. We consider only two: the Caesar cipher and random simple substitution cipher.

Outline

Introduction
- Basic Definitions
- Cryptanalysis
- Statistics of Natural Language

Transposition ciphers

Simple Substitution Ciphers
- Caesar Cipher
- Random Simple Substitution Cipher

Polyalphabetic Substitution
- Vigenère cipher
- Other polyalphabetic ciphers

Caesar cipher

- A cipher which moves the *i*th letter of an alphabet to the $(i + j)$th letter. The key is the value j.
- Instead of writing out the whole substitution table we can define encryption and decryption as follows.

$$\text{Encryption: } c_i = (a_i + j) \bmod n$$
$$\text{Decryption: } a_i = (c_i - j) \bmod n$$

where $n = 26$ or $n = 27$ (size of alphabet).

Example

If the key is $j = 1$ then CIPHER → DJQIFS

Cryptanalysis of Caesar cipher

- We only need to find where one of the most frequent characters is shifted to.
- Suppose that the ciphertext is:

  PACGHJUHHCRICGRFWRUCRICPHGLFLQH

  First count the characters. Most common characters are C and H with frequency of 5 each.

If we assume that ∇ is in the alphabet we just need to find where it is mapped to.

Trial 1: Try $\nabla \rightarrow$ H, i.e. $j = 8$.

HTVZ∇BM$\nabla\nabla$VJA...

not correct, since no recognisable words.

Trial 2: Try $\nabla \rightarrow$ C , i.e. j = 3

MY∇DEGREE∇OF∇DOCTOR∇OF∇MEDICINE

Outline

Introduction
- Basic Definitions
- Cryptanalysis
- Statistics of Natural Language

Transposition ciphers

Simple Substitution Ciphers
- Caesar Cipher
- Random Simple Substitution Cipher

Polyalphabetic Substitution
- Vigenère cipher
- Other polyalphabetic ciphers

Random simple substitution cipher

- A cipher which assigns a random character of the alphabet to another character of the alphabet.
- Encryption and decryption are defined by the substitution table which randomly permutes the alphabet.
- If the alphabet has 26 characters, there are 26! keys which is greater than 10^{26}. This is too many keys to search even with modern computers.
- The Caesar cipher is a special case of the random simple substitution cipher.

Example

Message: THE∇EVENING∇AND∇THE∇MORNING

Substitution table (key)		
A→S	J→G	S→M
B→J	K→C	T→O
C→V	L→F	U→Q
D→I	M→K	V→D
E→N	N→B	W→P
F→Y	O→U	X→ ∇
G→W	P→H	Y→T
H→A	Q→L	Z→X
I→Z	R→R	∇ →E

Message substitution (Encryption)		
T→O	N→B	D→I
H→A	I→Z	∇→E
E→N	N→B	T→O
∇→E	G→W	H→A
E→N	∇→E	E→N
V→D	A→S	∇→E
E→N	N→B	. . .

Ciphertext: OANENDNBZBWESBIEOANEKURBZBW

Cryptanalysis of random substitution

- Use frequency analysis on the characters of the alphabet.
- Decipher the following ciphertext:

```
FJLTXCFWKOVLHKJVKCBCOTEEVLPKCKJVJSTWTJYVKJVOJSTSBPLVITWCWPVDBIT
WICKTKQLVPHYTPRBJSTQLVYTKKJSCJETSCGTUHKJPTKYLFRTPETXCBTKJFXCJTJ
STGCZHTVOCGVZJCXTJTLCJCSHWPLTPOLCWYKFOJSTCQQCLCJHKVQTLCJTKEFJSV
HJCQQLTYFCRZTETCLJSTCXVLJFATXTWJKSVHZPRTYCZYHZCJTPCJCGTLBZVEOFI
HLTKCBQTLYTWJESFYSFKZCLITFWYVWJFWHVHKVQTLCJFVWFJEVHZPQLVPHYTXVL
TJSCWYHRFYXTJTLKVOICKCBTCLKCBCZFJJZTZTKKJSCWVWTYTWJFXTQTLYHRFYX
TJTLJSTYCHKJFYKVPCFKYVWKJCWJZBLTYHQTLCJTPCWPFKWTGTLPTKJLVBTPJST
KVZTQLVPHYJJSCJPFKCQQTCLKFKJSTPFKJFZZTPECJTLWVEVWTYHRFYXTJTLVOE
CJTLQLVPHYTKXVLTJSCWYHRFYXTJTLKVOICKJSTTDQTWKTFWECJTLJSTWPVTKWV
JCXVHWJJVCYTWJFXTQTLYHRFYXTJTLJSTILTCJOCYJVLVOJSTTDQTWKTLTKFPTK
FWJSTTZTYJLFYTWTLIBJSTYVKJVOKHLGTFZZCWYTEFZZRTXFWFXHXCWPJSTITWT
LCZTDQTWKTKCPZFRFJHX ...
```

Frequency analysis of ciphertext

No.	Character	%	Frequency
1	T	15.4	110
2	J	10.2	73
3	C	8.3	59
4	K	6.7	48
5	L	6.7	48
6	V	6.3	45

- Since E and T are most frequent characters in English we can guess the E → T and T → J in the substitution table.
- We can then start looking for English words like THE or other common trigrams.

Using Cryptool

- Solving random substitution by hand can be tedious and require a lot of trial and error
- We make use of software tools such as Cryptool (see link on course website)
- These tools can automate subtasks such as frequency counts or even automate the whole process

Key

With the help of tools we can find that the key (the substitution table) is:

Plaintext	a	b	c	d	e	f	g	h	i
Ciphertext	C	R	Y	P	T	O	I	S	F
Plaintext	j	k	l	m	n	o	p	q	r
Ciphertext	U	N	Z	X	W	V	Q	M	L
Plaintext	s	t	u	v	w	x	y	x	
Ciphertext	K	J	H	G	E	D	B	A	

The plaintext begins: `ITREMAINSFORUSTOSAY`...

Defining polyalphabetic substitution

- Polyalphabetic substitution ciphers use multiple mappings from plaintext to ciphertext.
- The effect of the multiple alphabets is to smooth the frequency distribution so direct frequency analysis is no longer effective.
- Typical polyalphabetic ciphers are periodic substitution ciphers based on a period *d*.
- Given *d* ciphertext alphabets $C_0, C_1, \ldots, C_{d-1}$, let

  $$f_i : A \to C_i$$

  be a mapping from the plaintext alphabet *A* to the *i*th cipher alphabet $C_i (0 \leq i \leq d - 1)$.

Encryption process

A plaintext message

$$M = m_0 \dots m_{d-1} m_d \dots m_{2d-1} \dots$$

is enciphered to

$$E(K, M) = f_0(m_0) \dots f_{d-1}(m_{d-1})\ f_0(m_d) \dots f_{d-1}(m_{2d-1}) \dots$$

For the special case $d = 1$ the cipher is monoalphabetic (a simple substitution cipher)

Random polyalphabetic substitution cipher

- Key Generation
  - Select block length d
  - Generate d random simple substitution tables
- Encryption
  - To encrypt the ith character, use the substitution table number j where $i \equiv j \pmod{d}$
- Decryption
  - Use the same substitution table as in encryption in order to reverse simple substitution

Example key for polyalphabetic substitution

Choose $d = 3$, so there are 3 ciphertext alphabets,

Key					
P:	abc	def	ghi	jkl	mno
C_1:	UWY	SX∇	TVZ	CEI	AFG
C_2:	QLM	PJO	RKN	∇XS	YUW
C_3:	MLQ	RNK	GFA	ZVT	YWU
P:	pqr	stu	vwx	yz∇	
C_1:	BDH	KNR	JOP	LMQ	
C_2:	ZVT	FGA	HDB	EIC	
C_3:	POJ	HDB	IEC	∇XS	

If P = IT∇IS∇A∇BEAUTIFUL∇DAY
then C = ZGSZFSUCLXQBNNKRSSSQ∇

Outline

- Introduction
  - Basic Definitions
  - Cryptanalysis
  - Statistics of Natural Language
- Transposition ciphers
- Simple Substitution Ciphers
  - Caesar Cipher
  - Random Simple Substitution Cipher
- Polyalphabetic Substitution
  - Vigenère cipher
  - Other polyalphabetic ciphers

Vigenère cipher

- The Vigenère cipher is a popular form of periodic substitution cipher based on *shifted* alphabets
- The key K is specified by a sequence of characters

$$K = k_0 \dots k_{d-1}$$

  where $k_i (i = 0, \dots, d-1)$ gives the amount of shift in the ith alphabet, i.e.

$$f_i(p) = (p + k_i) \bmod n$$

  where p is the plaintext character
- In the 19th century the Vigenère cipher was widely believed to be unbreakable

Example

M:	AT∇T	HE∇T	IME∇
K:	LOCK	LOCK	LOCK
$E(K, M)$:	LGBC	SSBC	T∇GJ

- We number the alphabet A=0, B=1 . . . Z=25, $\nabla = 26$
- In this example the first character of each 4-character group is shifted by 11, the second by 14, the third by 2 and the fourth by 10
- Shifting is computed modulo 27 so that the alphabet ‘wraps around’

Cryptanalysis of Vigenère cipher

1. Identify the period length. Several different techniques for this include:
   - Kasiski method (illustrated below)
   - Autocorrelation (used in Cryptool online version)
   - Index of Coincidence (used in JCryptool and Cyberchef)
2. Attack separately *d* different substitution tables. Since each substitution is just a shift (Caesar cipher), this is straightforward if there are sufficient ciphertexts.

Identifying the period using autocorrelation

- Given a ciphertext C compute the correlation between C and its shift C_i for all plausible values i of the period
- Because English is non-random, there is a better correlation between two texts with the same size shift than between two texts with different size shifts
- Therefore we expect to see peaks in the value of C_i when i is a multiple of the period
- Plotting the results on a histogram can usually allows us to identify the period
- This method can be used to find the period of any periodic polyalphabetic cipher

Example Vigenère cryptanalysis

The first characters of a ciphertext are:

AUVHSGE**PELPEK**QTEDKSFNYJYATCTCCKFTSUTEFVBVVHPNMFUHBFNPV
YFVRFVUSPEEVHFNAOFLBFYJPFPTMFFMFVHBVHFJAENEGVTIGHPWSFU
HPTTMAAGVESGIHJT**PELPEK**JPTIGMPTNJPGJUAUFOXPBFUIEGTIGFJTEIQ
WFXESYIUJTIGIOVEOVIPPOGCWBKTJPGIKMIQWFXESNOOIHFOIHJTCGIXC
SBNRFCDZFEFRLZKNUGRFUTFFIOJITKNRWISAFPTTIQUHJIUYATUUSTOVP
DFFBZPOOGOGVHFIRJOAOFSUTAOIEGGAUWRFUWIKCIYESGATUODKAUG
DXKTIVHFVWPERJOETYHJEHJJAWGAMTEBFYSGCPTDFFSUKLMVHFPAUW
RFQFUJEDCSFCNEVHFGXBNTFFSUCTJQNPHHJUCMKEOVGBXEJVADJASC
CUGRPHIUUOXPIOFEFFAQCRUHRPOTIGNBVUSGOGVHFKNWGSUKGBVIP
P**WIKCIO**YGTIFPDICDPPHBPDUJESGWBUSPOEUJIOIIOJITOATVESNYHTATR
OGCSJVUBVIPPAOFHJUKFGNJPCJUIWGRFCSPPIOI**WIKCIO**AEGIUCPMGAT
WRFVONGTPUTVFYIKSTASUGMPHWPTKBPDUQFPNLPYTIGQVKCLUUCVL
FOEUJOEUBZYHJEHIGDJUEOVAOILFFTIGMPUTJPEYVRJEACNENASUGRJ

Step 1

Identify the period length *d*.

- Note that the sequence PELPEK and WIKCIO occur multiple times.
- The positions of some of the pairs of these strings are separated by 117 and 93 characters.
- The period is almost certain to be 1 or 3 because the only common divisors of 117 and 93 are 1 and 3.

This process is known as the Kasiski method. We can also automate the process by plotting the autocorrelation (use CrypTool).

Step 2

- Attack separately three different alphabets
- Only need to find the shift for each alphabet as in Caesar cipher
- Look for character with largest frequency and assume this is shifted from E.
- Turns out that:
  - the first has key ‘A’ (shift of 0).
  - the second has key ‘B’ (shift of 1), and
  - the third has key ‘C’ (shift of 2).

The plaintext starts:

`ATTHREEOCLOCKPRECISELYIWASATBAKERSTREET...`

Outline

- Introduction
  - Basic Definitions
  - Cryptanalysis
  - Statistics of Natural Language
- Transposition ciphers
- Simple Substitution Ciphers
  - Caesar Cipher
  - Random Simple Substitution Cipher
- Polyalphabetic Substitution
  - Vigenère cipher
  - Other polyalphabetic ciphers

Other ciphers designed for use by hand

You can find many other ciphers in the tools such as Cryptool and Cyberchef. Some examples:

- the *autokey cipher* starts off as the Vigenère cipher but once the alphabets defined by the key have been used once, uses the plaintext to define subsequent alphabets. Therefore the autokey cipher is *not* periodic.
- the *running key cipher* uses a (practically) infinite set of alphabets from a shared key. In practice the shared key can be extract from a book, when it is often called a *book cipher*.

Rotor machines

- In the early 20th century electromechanical machines were developed for encryption using *rotors* as moving alphabets.
- The most famous is the Enigma machine used by the Germans in World War II.
- Each character is encrypted using a different alphabet. The Enigma machine has a period of about 17000, so in practice it would never repeat on the same message.
- Smart's book (see the additional resources list) has a whole chapter on Enigma and how it was broken.

Lecture 4: Hill Cipher, Stream Ciphers and the One Time Pad

TTM4135

Relates to Stallings Chapter 3

Spring Semester, 2025

Motivation

- The Hill Cipher is a mathematically defined encryption scheme
- The Hill Cipher illustrates the weakness of linearity in cipher design
- Stream ciphers are constructed from (pseudo-)random number generators.
- The One Time Pad is an unbreakable stream cipher

Outline

Hill Cipher

Stream ciphers

The One Time Pad

Visual Cryptography

Hill cipher

- Lester S. Hill was an American mathematician who published his cipher in 1929.
- The Hill cipher is an example of a *polygram cipher* (also called *polygraphic cipher*). This is a simple substitution cipher on an extended alphabet consisting of multiple characters. The simplest example is digram substitution in which the alphabet consists of all pairs of characters.
- The major weakness of the Hill cipher is that it is linear. This makes known plaintext attacks easy.

Definition of Hill cipher

The Hill cipher performs a linear transformation on d plaintext characters to get d ciphertext characters.

- Encryption involves multiplying a $d \times d$ matrix K by the block of plaintext P.
- Decryption involves multiplying the matrix K^{-1} by the block of the ciphertext C.

$$\begin{array}{ll} \text{Encryption:} & C = KP \\ \text{Decryption:} & P = K^{-1}C \end{array}$$

Encryption example

- We choose $d = 2$ so that encryption takes digrams as input and output blocks
- Write each plaintext pair as a column vector and encode letters as numbers
- Suppose the first pair for encryption is (EG). Then since E=4 and G=6 in our encoding this is represented as $\begin{pmatrix} 4 \\ 6 \end{pmatrix}$
- If there are insufficient letters to fill a block then it must be padded. This can be done with an uncommon letter such as Z
- In these examples the space character is omitted and all computations take place modulo 26

Encrypting and decrypting

$$d = 2, \quad K = \begin{pmatrix} 4 & 5 \\ 1 & 7 \end{pmatrix}, \quad K^{-1} = \begin{pmatrix} 15 & 19 \\ 9 & 16 \end{pmatrix}$$

$$Plaintext : (BC) \rightarrow P = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

$$Encryption : C = KP = \begin{pmatrix} 4 & 5 \\ 1 & 7 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 14 \\ 15 \end{pmatrix} \rightarrow (OP)$$

$$Decryption : P = K^{-1}C = \begin{pmatrix} 15 & 19 \\ 9 & 16 \end{pmatrix} \begin{pmatrix} 14 \\ 15 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Cryptanalysis of Hill cipher

- Known plaintext attack is possible given d plaintext-ciphertext matching blocks.
- Suppose we are given blocks (column vectors) P_i, C_i for $i = 0, 1, .., d-1$.
  1. Let $C = [C_0\, C_1\, \ldots\, C_{d-1}]$. Let $P = [P_0\, P_1\, \ldots\, P_{d-1}]$.
  2. Solve $C = KP$ for K.
  3. $P = K^{-1}C$.

Cryptanalysis example

- Suppose that we know $d = 2$.
- Ciphertext: ZIKPWIXPTFUTVPVRQBUTVPJLKB
- Known plaintext is first two blocks (digrams): THER

Step 1 - encode plaintext and ciphertext

$$P_0 = (TH) = \begin{pmatrix} 19 \\ 7 \end{pmatrix},\ P_1 = (ER) = \begin{pmatrix} 4 \\ 17 \end{pmatrix}$$

$$C_0 = (ZI) = \begin{pmatrix} 25 \\ 8 \end{pmatrix},\ C_1 = (KP) = \begin{pmatrix} 10 \\ 15 \end{pmatrix}$$

$$\to P = [P_0\ P_1] = \begin{pmatrix} 19 & 4 \\ 7 & 17 \end{pmatrix}$$

$$\to C = [C_0\ C_1] = \begin{pmatrix} 25 & 10 \\ 8 & 15 \end{pmatrix}$$

Step 2 - recover encryption matrix K

We have $C = KP$. Let $K = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. Then:

$$\begin{pmatrix} 25 & 10 \\ 8 & 15 \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} 19 & 4 \\ 7 & 17 \end{pmatrix}.$$

So

$$\begin{aligned} \begin{pmatrix} a & b \\ c & d \end{pmatrix} &= \begin{pmatrix} 25 & 10 \\ 8 & 15 \end{pmatrix} \begin{pmatrix} 19 & 4 \\ 7 & 17 \end{pmatrix}^{-1} \\ &= \begin{pmatrix} 25 & 10 \\ 8 & 15 \end{pmatrix} \begin{pmatrix} 25 & 14 \\ 5 & 5 \end{pmatrix} \\ &= \begin{pmatrix} 25 & 10 \\ 15 & 5 \end{pmatrix} \end{aligned}$$

.

Step 3 - compute K^{-1} and decrypt ciphertext

$$\rightarrow K = \begin{pmatrix} 25 & 10 \\ 15 & 5 \end{pmatrix} \rightarrow \cdots \rightarrow K^{-1} = \begin{pmatrix} 5 & 16 \\ 11 & 25 \end{pmatrix}$$

$$\begin{aligned} C &= \begin{pmatrix} W & X & T & U & T & \dots \\ I & P & F & T & V & \dots \end{pmatrix} \\ &= \begin{pmatrix} 22 & 23 & 17 & 20 & 17 & \dots \\ 8 & 15 & 5 & 17 & 21 & \dots \end{pmatrix} \end{aligned}$$

$$P = K^{-1} C$$

Plaintext: THEREARETWOTHINGSTOTHINKOF

Notes on cryptanalysis of Hill cipher

- In known plaintext attacks the equations may not be fully determined. In this case Step 2 will fail because the matrix will not be invertible. Further plaintext/ciphertext character can be examined.
- Ciphertext only attack follows known plaintext attack with the added task of finding probable blocks of matching plaintext and ciphertext. For example, when $d = 2$ the frequency distribution of non-overlapping pairs of ciphertext characters can be compared with the distribution of pairs of plaintext characters.

Stream ciphers

- Stream ciphers are characterised by the generation of a *keystream* of any required length
- Each element of the keystream is used successively to encrypt one or more ciphertext characters
- Stream ciphers are usually symmetric key ciphers: sender and receiver share the same key and can generate the same keystream given the same initialisation value
- The keystream must have good randomness properties

Synchronous stream ciphers

- In the simplest kind of stream cipher the keystream is generated *independently* of the plaintext. In this case the cipher is called a *synchronous* stream cipher.
- Both sender and receiver need to generate the same keystream and synchronise on its usage
- The Vigenère cipher can be seen as a (periodic) synchronous stream cipher where each shift is defined by a key letter
- Later we will see how to use modern block ciphers to generate a keystream

Binary synchronous stream cipher

For each time interval t each of the following are defined:

- a binary sequence $s(t)$ called the *keystream*;
- a binary plaintext $p(t)$;
- a binary ciphertext $c(t)$.

$$\begin{array}{ll} \text{Encryption:} & c(t) = p(t) \oplus s(t) \\ \text{Decryption:} & p(t) = c(t) \oplus s(t) \end{array}$$

One-time pad

- Often attributed to Vernam who made a one-time pad machine using teletype machinery in 1917. Earlier historical uses are known.
- The key is a truly random sequence of characters, all of them independently generated
- Each character in the key is used *one time* only
- The alphabet can be of any length, but usually is either a natural language alphabet or simply the binary alphabet $\{0, 1\}$.
- The binary one time pad is a (non-periodic) binary synchronous stream cipher.
- The one-time pad provides perfect secrecy.

Shannon's definition of perfect secrecy

- To define perfect secrecy, consider a cipher with message set $\{M_1, M_2, \ldots, M_k\}$ and ciphertext set $\{C_1, C_2, \ldots, C_l\}$.
- Then $\Pr(M_i|C_j)$ is the probability that message M_i was encrypted given that ciphertext C_j was observed.
- Note that in most cases the messages M_i will *not* be equally likely.
- We say that the cipher achieves *perfect secrecy* if for all messages M_i and ciphertexts C_j we have

$$\Pr(M_i|C_j) = \Pr(M_i)$$

One time pad using Roman alphabet

- Plaintext characters: $p_1, \ldots, p_r$
- Ciphertext characters: $c_1, \ldots, c_r$
- Keystream: random characters $k_1, \ldots, k_r$
- Encryption:
$$c_i = (p_i + k_i) \bmod 26$$
- Decryption:
$$p_i = (c_i - k_i) \bmod 26$$
- Resulting ciphertext is modulo 26 addition of the plaintext and keystream sequences.

Why the one time pad provides perfect secrecy

- Suppose a particular ciphertext C_j is observed.
- Any message could have been sent depending on the choice of key.
- The probability that message M_i was sent given that C_j is observed is the probability that M_i is chosen, weighted by the probability that the right key was chosen.
- Since each key is chosen with equal probability, the conditional probability $\Pr(M_i|C_j)$ is simply $\Pr(M_i)$.

Example

Plaintext:	HELLO
Keystream:	EZABD
Ciphertext:	LDLMR

- Note that given the ciphertext LDLMR the plaintext can be *any* 5-letter message.

Real one-time pads used by spies in 1960s

Vernam (binary) one time pad

- Plaintext is binary sequence: $b_1, b_2, \ldots, b_r$
- Keystream is random binary sequence: $k_1, k_2, \ldots, k_r$
- Ciphertext is binary sequence: $c_1, c_2, \ldots, c_r$
- Encryption: $c_i \equiv p_i \oplus k_i$
- Decryption: $p_i \equiv c_i \oplus k_i$
- Keystream is same length as plaintext
- Provides perfect secrecy since any ciphertext is equally possible given the plaintext
- Encryption and decryption are identical processes

One-time pad properties

- Shannon showed that any cipher with perfect secrecy *must* have as many keys as there are messages.
- In this sense the one-time pad is the only unbreakable cipher.
- Practical usage is possible for pre-assigned communications between fixed parties.
- Main problem with one time pad as a general tool is how to deal with key management of completely random keys.
- Key generation, key transportation, key synchronization, key destruction are all problematic since the keys are so large.
- In Caesar cipher, the key legth was one integer between $\{0, \ldots, 26\}$ ($\approx$ 5 bits). Now, the key is the length of the message.

Key management issues for one time pad

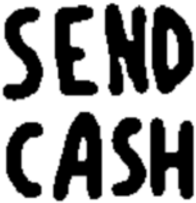
- How to generate completely random keys?
- How to transport random keys between sender and receiver?
- How to synchronise on usage of keys?
- How to destroy keys after use?

Visual cryptography

- A fun application of the one time pad is *visual cryptography* which splits an image into two *shares*
- Decryption works by *overlaying* the two shared images
- First proposed by Naor and Shamir in 1994
- We consider the simplest case of monochrome images with black or white pixels — many generalisations are possible
- Each share reveals no information about the image - this is unconditional security as in the one time pad

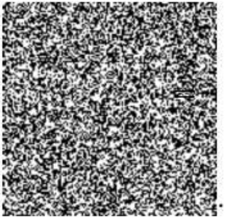
Two-time Pad

Message:



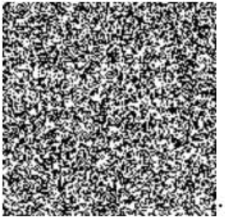
⊕

OTP key:



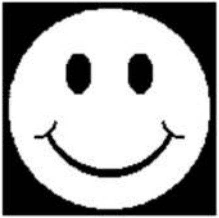
=

Ciphertext:

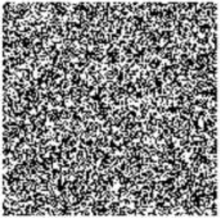


Two-time Pad

Message:

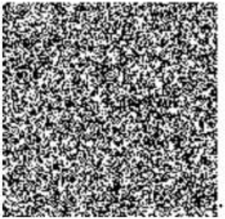


⊕

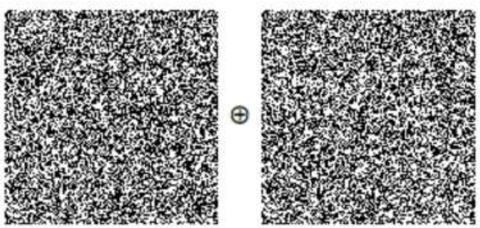


Ciphertext:

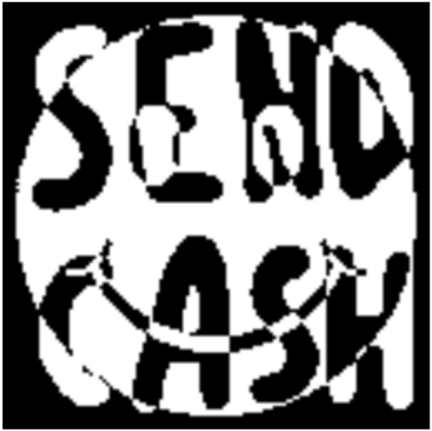
=



Two-time Pad

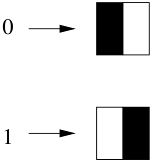


Two-time Pad



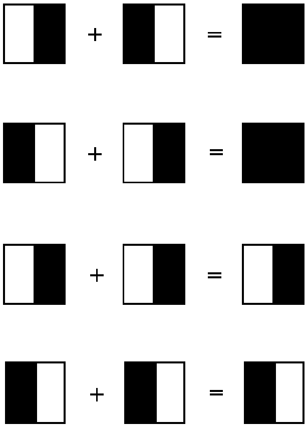
Encrypting in visual cryptography

- To encrypt an image I, first generate a one time pad P (random string of bits) with length equal to the number of pixels of I
- Generate an image share S_1 by replacing each bit in P using the sub-pixel patterns shown
- Generate the other image share S_2 with pixels as follows:
  - the same as S_1 for all the white pixels of I
  - the opposite (other sub-pixel pattern) of S_1 for all the black pixels of I



Decrypting in visual cryptography

- To reveal the hidden image the two shares are overlayed
- Each black pixel of I is black in the overlay
- Each white pixel of I is half white in the overlay



Lecture 5: Block Ciphers

TTM4135

Relates to Stallings Chapter 4 and 6

Spring Semester, 2025

Motivation

- Block ciphers are the main bulk encryption algorithms used in commercial applications.
- Standardised block cipher AES and legacy cipher DES are widely deployed in real applications.
- NIST's AES algorithm validation list includes over 13500 validated implementations including examples such as USB drives, door controllers, media server encryption, disk encryption, bluetooth devices, iPhone and hundreds more.

Outline

- Block Cipher Principles
    - Product Ciphers and Iterated Ciphers
    - Feistel Ciphers
    - Substitution-permutation networks
    - Standard security properties
- DES
    - History of DES
    - DES algorithm
    - Brute Force Attack on DES
    - Double and triple DES
- AES
    - AES History
    - AES Algorithm
    - Comparison of AES and DES

Block ciphers

- Block ciphers are symmetric key ciphers in which each block of plaintext is encrypted with the same key.
- A *block* is a set of plaintext symbols of a fixed size. Typical block sizes for modern block ciphers are between 64 and 256 bits.
- In practice block ciphers are used in certain configurations called *modes of operation*. We look at modes in a later lecture.

Notation in this lecture

- P: Plaintext block (length n bits)
- C: Ciphertext block (length n bits)
- K: Key (length k bits)
- $C = E(P, K)$: Encryption function
- $P = D(C, K)$: Decryption function

Criteria for block cipher design

In the 1940s Claude Shannon discussed two important encryption techniques.

- **Confusion**: This involves substitution to make the relationship between the key and ciphertext as complex as possible.
- **Diffusion**: This involves transformations that dissipate the statistical properties of the plaintext across the ciphertext.

Shannon proposed to use these techniques repeatedly using the concept of *product cipher*.

Outline

- Block Cipher Principles
  - Product Ciphers and Iterated Ciphers
  - Feistel Ciphers
  - Substitution-permutation networks
  - Standard security properties
- DES
  - History of DES
  - DES algorithm
  - Brute Force Attack on DES
  - Double and triple DES
- AES
  - AES History
  - AES Algorithm
  - Comparison of AES and DES

Product cipher

- A product cipher is a cryptosystem in which the encryption function is formed by applying (or *composing*) several sub-encryption functions.
- Most block ciphers are the composition of simple functions f_i for $i = 1, \ldots, r$ where each f_i has a different key K_i.
- Thus we can write

$$C = E(P, K) = f_r(\ldots(f_2(f_1(P, K_1), K_2)\ldots), K_r)$$

Iterated ciphers

Most modern block ciphers are in a special class of product ciphers known as *iterated ciphers*.

- The encryption process is divided into r similar *rounds*
- The sub-encryption functions are all the same function, g, called the *round function*
- Each key K_i is derived from the overall master key K. The keys K_i are called *round keys* or *subkeys* and are derived from K using a process called the *key schedule*

Encryption in iterated ciphers

Given a plaintext block, P, a round function g and round keys K_1, K_2, . . . ,K_r, the ciphertext block, C, is derived through r rounds as follows.

$$\begin{aligned} W_0 &= P \\ W_1 &= g(W_0, K_1) \\ W_2 &= g(W_1, K_2) \\ \vdots \; & \vdots \; \vdots \\ W_r &= g(W_{r-1}, K_r) \\ C &= W_r \end{aligned}$$

Decrypting iterated ciphers

- The round function g must have an inverse function g^{-1} with $g^{-1}(g(W, K_i), K_i) = W$ for all keys K_i and blocks W.
- Decryption is then the reverse of encryption.

$$\begin{aligned} W_r &= C \\ W_{r-1} &= g^{-1}(W_r, K_r) \\ W_{r-2} &= g^{-1}(W_{r-1}, K_{r-1}) \\ \vdots \; &\; \vdots \; \vdots \\ W_0 &= g^{-1}(W_1, K_1) \\ P &= W_0 \end{aligned}$$

Types of iterated cipher

Two widely used general block cipher designs are:

1. **Feistel ciphers:** an example is the Data Encryption Standard (DES)
2. **Substitution-Permutation Networks (SPNs):** an example is the Advanced Encryption Standard (AES)

Outline

- Block Cipher Principles
  - Product Ciphers and Iterated Ciphers
  - Feistel Ciphers
  - Substitution-permutation networks
  - Standard security properties
- DES
  - History of DES
  - DES algorithm
  - Brute Force Attack on DES
  - Double and triple DES
- AES
  - AES History
  - AES Algorithm
  - Comparison of AES and DES

Feistel cipher

- Named after Horst Feistel, a cryptographer working for IBM who influenced the design of DES
- A Feistel cipher is an iterated cipher in which the round function swaps the two halves of the block and forms a new right hand half
- The process is sometimes called a *Feistel network* since the process can be seen as a network which the two halves of the plaintext block travel through.

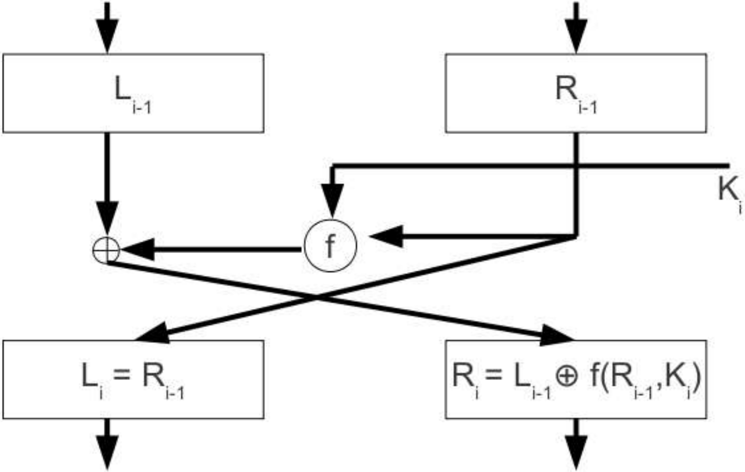
Feistel encryption

1. Split plaintext block $P = W_0$ into two halves: $W_0 = (L_0, R_0)$.
2. For each of the r rounds perform the following:

$$
\begin{aligned}
L_i &= R_{i-1} \\
R_i &= L_{i-1} \oplus f(R_{i-1}, K_i)
\end{aligned}
$$

3. Ciphertext $C = W_r$ is defined by $C = (L_r, R_r)$.

Feistel ladder diagram



Feistel ladder diagram

Source: Introduction to Modern Cryptography, Jonathan Katz and Yehuda Lindell, third edition

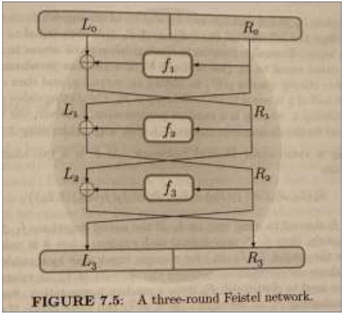


FIGURE 7.5: A three-round Feistel network.

Feistel decryption

1. Write the ciphertext block C as $C = (L_r, R_r)$.
2. For each of the r rounds perform the following:

$$\begin{aligned} L_{i-1} &= R_i \oplus f(L_i, K_i) \\ R_{i-1} &= L_i \end{aligned}$$

3. Finally the plaintext is $P = (L_0, R_0)$

- We never have to invert the function f so we can always decrypt for *any* function f.
- However, choice of f is critical for security as it is the only **non-linear** part of the encryption function.

Outline

- Block Cipher Principles
  - Product Ciphers and Iterated Ciphers
  - Feistel Ciphers
  - Substitution-permutation networks
  - Standard security properties
- DES
  - History of DES
  - DES algorithm
  - Brute Force Attack on DES
  - Double and triple DES
- AES
  - AES History
  - AES Algorithm
  - Comparison of AES and DES

SPNs

- A substitution-permutation network is an iterated cipher.
- The block length n must allow each block to be split into m sub-blocks of length l so that $n = lm$. Two permutations are defined.
- Permutation π_S operates on sub-blocks of size l bits:

$$\pi_S : \{0,1\}^l \rightarrow \{0,1\}^l$$

  The permutation π_S is normally called an S-box (substitution box).
- Permutation π_P swaps the inputs from $\{1, \ldots, n\}$. This is similar to the transposition cipher.

$$\pi_P : \{1,2,\ldots,n\} \rightarrow \{1,2,\ldots,n\}$$

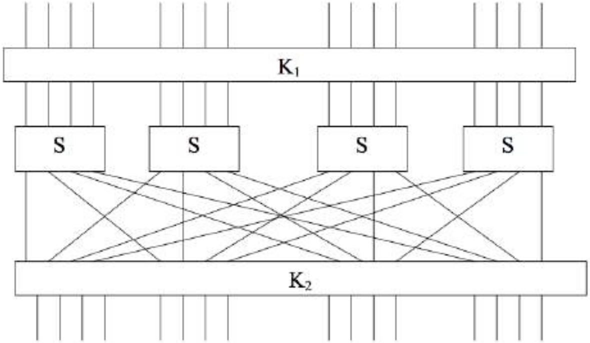
Steps in SPN round function

The round function is defined by three steps

1. The round key K_i is XORd with the current state block W_i
2. Each sub-block is replaced (substituted) by application of π_S
3. The whole block is permuted using π_P.

In the following picture the boxes marked S implement the permutation π_S. One complete round is shown with the start of a second one.

SPN network



- The round key K_i is added (XOR) into the current block
- The same substitution, S, is applied to each sub-block
- The whole block is permuted at the bit level (transposition)

One-round SPN

Source: Introduction to Modern Cryptography, Jonathan Katz and Yehuda Lindell, third edition

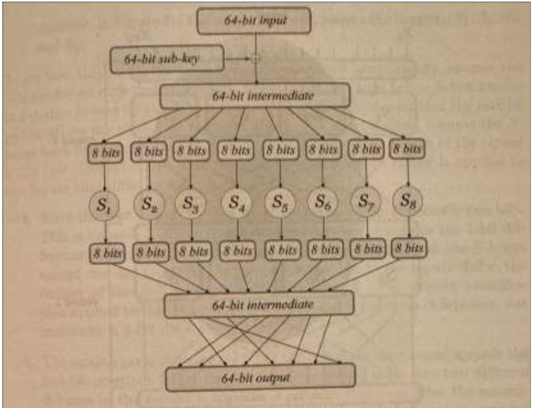


FIGURE 7.3: A single round of a substitution-permutation network.

Three-round SPN

Source: Introduction to Modern Cryptography, Jonathan Katz and Yehuda Lindell, third edition

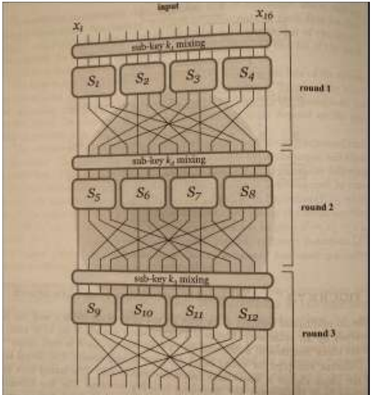


FIGURE 7.4: Three rounds of a substitution-permutation network

Outline

- Block Cipher Principles
  - Product Ciphers and Iterated Ciphers
  - Feistel Ciphers
  - Substitution-permutation networks
  - Standard security properties
- DES
  - History of DES
  - DES algorithm
  - Brute Force Attack on DES
  - Double and triple DES
- AES
  - AES History
  - AES Algorithm
  - Comparison of AES and DES

Avalanche effects

- Good block ciphers typically exhibit *avalanche effects* with respect to both key and plaintext.
- **Plaintext avalanche**: a small change in the plaintext should result in a large change in the resulting ciphertext. Ideally, changing one bit of the plaintext changes each of the bits in the output block with probability 1/2.
- We can relate the plaintext avalanche effect to Shannon's notion of diffusion.
- **Key avalanche**: a small change in the key (with the same plaintext) should result in a large change in the resulting ciphertext.
- We can relate the key avalanche effect to Shannon's notion of confusion.

Differential and Linear Cryptanalysis

- Differential cryptanalysis is a powerful technique first published in 1992. It is a chosen plaintext attack.
- Based on the idea that the difference between two input plaintexts can be correlated to the difference between two output ciphertexts.
- Linear cryptanalysis is a known plaintext attack first published in 1993. It can be theoretically used to break DES.
- Modern block ciphers are normally designed to be immune to both differential and linear cryptanalysis.

Outline

- Block Cipher Principles
  - Product Ciphers and Iterated Ciphers
  - Feistel Ciphers
  - Substitution-permutation networks
  - Standard security properties
- DES
  - History of DES
  - DES algorithm
  - Brute Force Attack on DES
  - Double and triple DES
- AES
  - AES History
  - AES Algorithm
  - Comparison of AES and DES

Data Encryption Standard (DES)

- Designed by researchers from IBM and submitted to the NBS (National Bureau of Standards) in US in a call for a publicly available cipher.
- Approved in 1977 as the US standard for encryption.
- The encryption and decryption definitions are public property. The security of the DES algorithm resides in the difficulty of decryption without knowledge of the key.
- DES is a 16-round Feistel cipher with key length of 56 bits and data block length of 64 bits.

Outline

- Block Cipher Principles
  - Product Ciphers and Iterated Ciphers
  - Feistel Ciphers
  - Substitution-permutation networks
  - Standard security properties
- DES
  - History of DES
  - DES algorithm
  - Brute Force Attack on DES
  - Double and triple DES
- AES
  - AES History
  - AES Algorithm
  - Comparison of AES and DES

Encryption operation

An input block of 64 bits denoted by P.

Step 1 The 64 bits of P are permuted according to an initial fixed permutation, denoted by IP.

Step 2 After the permutation, 16 rounds of a Feistel operation are applied, denoted by function f. A different 48 bit subkey is used for each round of the f function.

Step 3 After the 16 round operations, a final fixed inverse permutation, denoted by IP^{-1}, is applied.

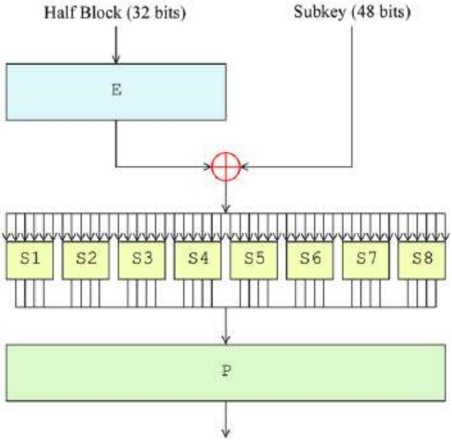
After Step 3, the output ciphertext block, denoted by C, has been formed.

DES Feistel operation

For each round the following steps are followed (see picture on next slide)

- Step 1 Expand 32 bits to 48 bits
- Step 2 Bitwise modulo two add (XOR) 48 bits to 48 bit subkey for round
- Step 3 Break 48 bits into eight blocks of six bits each
- Step 4 Put block *i* into substitution table *i* resulting in block of length four.
- Step 5 Apply permutation to resulting 32 bits.

Feistel f function used in DES



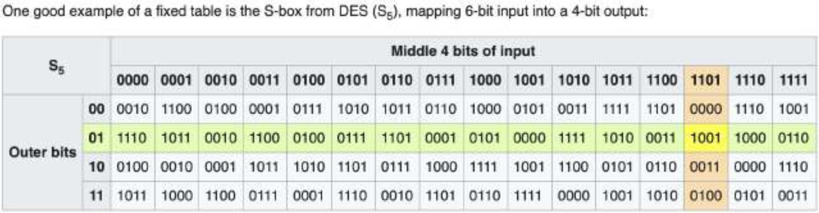
Picture courtesy of Wikimedia commons

S-box example I

Row	Column Number															
No.	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

- Suppose input block B is $x_1x_2x_3x_4x_5x_6$
- Digits x_1 and x_6 define row number between 0 and 3
- Digits $x_2x_3x_4x_5$ define column number between 0 and 15

S-box example II



One good example of a fixed table is the S-box from DES (S_5), mapping 6-bit input into a 4-bit output:

S_5		Middle 4 bits of input															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

https://en.wikipedia.org/wiki/S-box

Key schedule

- Each of the sixteen rounds involves 48 bits of the 56 bit key.
- Each 48-bit subkey is defined by a series of permutations and shifts on the full 56-bit key

Outline

- Block Cipher Principles
  - Product Ciphers and Iterated Ciphers
  - Feistel Ciphers
  - Substitution-permutation networks
  - Standard security properties
- DES
  - History of DES
  - DES algorithm
  - Brute Force Attack on DES
  - Double and triple DES
- AES
  - AES History
  - AES Algorithm
  - Comparison of AES and DES

Brute force attack

- A brute force attack on a block cipher consists of testing all possible 2^k keys in order to find the key K.
- The right key can be identified by using a small number of ciphertext blocks, or by looking for low entropy in the decrypted plaintext.
- In the case of DES there are 2^{56} keys to test so that, on the average, it would take 2^{55} trial samples to find the right key.
- Right from its first publication the short size of the DES key was criticised (by academics).
- As technology evolved (i.e. computational power), this became insecure.

Real world attacks

1997	• $10,000 DES Challenge in February 1997 • Solved in June 1997 • Linked together thousands of computers over the Internet (parallel processing)
1998	• EFF DES Cracker built • cost less than $250 000 • less than three days to find 56-bit DES key • searched 88 billion keys per second
1999	• EFF DES Cracker plus distributed search • 22 hours 15 minutes to find 56-bit DES key • searched 245 billion keys per second
2007	• Parallel FPGA-based machine COPACOBANA • cost $10,000 to build • less than 1 week to find 56-bit DES key

Outline

- Block Cipher Principles
  - Product Ciphers and Iterated Ciphers
  - Feistel Ciphers
  - Substitution-permutation networks
  - Standard security properties
- DES
  - History of DES
  - DES algorithm
  - Brute Force Attack on DES
  - Double and triple DES
- AES
  - AES History
  - AES Algorithm
  - Comparison of AES and DES

Double encryption

- Let K_1 and K_2 denote two keys of the block cipher. Then double encryption is defined by:

$$C = E(E(P, K_1), K_2) \tag{1}$$

- If the key length of the original block cipher is k then exhaustive key attack requires 2^{2k-1} trials on average.
- In fact there is a time-memory tradeoff which reduces this using a meet-in-the-middle method.

Meet-in-the-middle attack on double encryption

Suppose we have a single ciphertext/plaintext pair (P, C) satisfying equation (1).

Step 1. For each key, store $C' = E(P, K)$ in memory.

Step 2. Check whether $D(C, K') = C'$ for any key K'.

Step 3. K from Step 1 is K_1 and K' from Step 2 is K_2.

Step 4. Check whether key values in Step 3 work for other (P, C) pairs.

This attack requires storage of one plaintext block for every possible key.

Attack applied to double DES

- The attack requires:
  - storage of one plaintext block for every key
  - a single encryption for every key
  - a single decryption for every key
- For DES algorithm this would require storage of 2^{56} 64-bit blocks, 2^{56} encryption operations and 2^{56} decryption operations.
- Expensive, but much easier than brute force search through 2^{111} keys.

Triple encryption

- Much better security can be provided by using triple encryption.
- In general three keys K_1, K_2 and K_3 are used. Then encryption is defined by:

$$C = E(D(E(P, K_1), K_2), K_3)$$

- This is secure from the above meet-in-the-middle attack.
    - *EDE* for backward compatibility (if the keys are the same, i.e. $K_1 = K_2 = K_3 = K$ this is the same as E)
    - If E has strong pseudorandomn properties, so does $D = E^{-1}$.

Standardised options

- The 1999 version of the DES standard specified three options.
  1. Use three independent keys K_1, K_2, K_3. The most secure.
  2. Use two keys with $K_1 = K_3$. Still secure enough.
  3. Use one key with $K_1 = K_2 = K_3$. Backward compatible with single key DES (vulnerable to brute-force key search).
- NIST SP 800-131A, March 2019 states:
  - Two-key triple DES is allowed only for *legacy use* (decryption only).
  - Three-key triple DES remains allowed in existing applications only, and after 2023 only for legacy use.

Outline

- Block Cipher Principles
  - Product Ciphers and Iterated Ciphers
  - Feistel Ciphers
  - Substitution-permutation networks
  - Standard security properties
- DES
  - History of DES
  - DES algorithm
  - Brute Force Attack on DES
  - Double and triple DES
- AES
  - AES History
  - AES Algorithm
  - Comparison of AES and DES

Advanced Encryption Standard (AES)

- Due to controversy over DES design, AES was designed in an open competition
- Process took several years with much public debate
- From 15 original submissions, 5 finalists were all widely believed secure
- Winner was Rijndael, designed by two Belgian cryptographers, Vincent Rijmen and Joan Daeman

Outline

- Block Cipher Principles
  - Product Ciphers and Iterated Ciphers
  - Feistel Ciphers
  - Substitution-permutation networks
  - Standard security properties
- DES
  - History of DES
  - DES algorithm
  - Brute Force Attack on DES
  - Double and triple DES
- AES
  - AES History
  - AES Algorithm
  - Comparison of AES and DES

AES overview

- Symmetric key block cipher
- 128-bit data block; 128-, 192- or 256-bit master key
- Number of rounds, NR, is 10, 12 or 14 (for 128-, 192-, 256-bit keys)
- Byte-based design
- Structure is essentially a substitution-permutation network:
    - initial round key addition
    - NR-1 rounds
    - final round

State - matrix of bytes

Data block size = 16 bytes

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

- byte-based
- mixture of finite field operations in $GF(2^8)$ and bit string operations.

Round transformation

Four basic operations:

1. ByteSub (non-linear substitution)
2. ShiftRow (permutation)
3. MixColumn (diffusion)
4. AddRoundKey

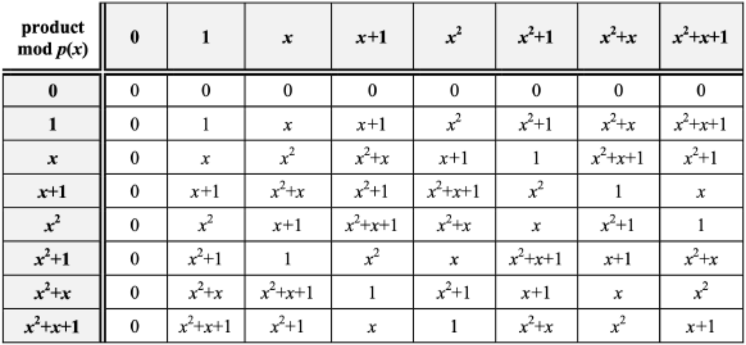
- Essentially a substitution-permutation network with $n = 128$ and $l = 8$
- S-box is look-up table but mathematically defined in $GF(2^8)$
- CrypTool allows step-by-step computation of the encryption and decryption process

Example

Reminder on blackboard.

GF(8) example

The polynomial $x^3 + x + 1$ is irreducible in $\mathbb{Z}_2$.



product mod $p(x)$	**0**	**1**	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
0	0	0	0	0	0	0	0	0
1	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
x	0	x	x^2	x^2+x	$x+1$	1	x^2+x+1	x^2+1
$x+1$	0	$x+1$	x^2+x	x^2+1	x^2+x+1	x^2	1	x
x^2	0	x^2	$x+1$	x^2+x+1	x^2+x	x	x^2+1	1
x^2+1	0	x^2+1	1	x^2	x	x^2+x+1	$x+1$	x^2+x
x^2+x	0	x^2+x	x^2+x+1	1	x^2+1	$x+1$	x	x^2
x^2+x+1	0	x^2+x+1	x^2+1	x	1	x^2+x	x^2	$x+1$

http://homepages.math.uic.edu/~leon/mcs425-s08/handouts/field.pdf

Key schedule

- The master key input is 128 bits (or 192 bits or 256 bits).
- Each of the 10 (or 12 or 14 respectively) encryption/decryption rounds uses a 128-bit subkey.
- The number of subkeys required is one for each round (10 or 12 or 14) plus an initial subkey. Therefore, for a 128-bit key 11 subkeys are required.
- The key schedule derives the eleven 128-bit subkeys from the 128-bit master key.

AES security

- Some cracks have appeared but not significant breaks
- Attacks exist on reduced-round versions
- *Related key attacks* exist. Such attacks require the attacker to obtain ciphertext encrypted with a key related to the actual key in a specified way.
- Most serious real attack so far reduces effective key size by around 2 bits.

Outline

- Block Cipher Principles
  - Product Ciphers and Iterated Ciphers
  - Feistel Ciphers
  - Substitution-permutation networks
  - Standard security properties
- DES
  - History of DES
  - DES algorithm
  - Brute Force Attack on DES
  - Double and triple DES
- AES
  - AES History
  - AES Algorithm
  - Comparison of AES and DES

DES/AES comparison

- **Data block size**: DES - 64 bits; AES - 128 bits
- **Key size**: DES - 56 bits; AES - 128, 192 or 256 bits
- **Design structure**:
  - both are iterated ciphers
  - DES has a Feistel structure; AES is a SPN;
  - DES is bit-based; AES is byte-based
  - AES substantially faster in both hardware and software

Conclusion

- Block ciphers are the workhorses of secure communications
- AES is the choice of today but triple-DES is still in use in older applications
- Designing good block ciphers is a difficult and time-consuming process and requires years of validation by experts
- In future lectures we will see how to use block ciphers as a building block for confidentiality and authentication

Lecture 6: Modes of Operation and Random Numbers

TTM4135

Relates to Stallings Chapters 7 and 8

Spring Semester, 2025

Motivation

- Block ciphers encrypt single blocks of data but in applications many blocks of data are encrypted sequentially
- The simple approach to break up the plaintext into blocks and encrypt each separately is generally insecure
- There are many different *modes of operation* which are standardised with different security and efficiency
- Random numbers are needed in many uses of cryptography
- Block ciphers can be used to generate random numbers

Outline

- Important Features of Different Modes
- Standards
- Confidentiality Modes
  - Electronic Codebook (ECB) Mode
  - Cipher Block Chaining (CBC) Mode
  - Cipher Feedback (CFB) Mode
  - Output Feedback (OFB) Mode
  - Counter (CTR) Mode
- Random numbers
  - DRBGs
  - CTR_DRBG
  - Dual_EC_DRBG
- An example: Cloudfare

Purpose of different modes

- Modes can be designed to provide *confidentiality* for data, or to provide *authentication* (and integrity) for data or to provide both
- In this lecture we focus on **confidentiality** modes which normally must include **randomisation**
- Some modes can be used to generate pseudorandom numbers
- Different modes have different efficiency properties or communications properties

Reminder from Lecture 3: Vigenère cryptanalysis

The first characters of a ciphertext are:

AUVHSGE**PELPEK**QTEDKSFNYJYATCTCCKFTSUTEFVBVVHPNMFUHBFNPV

YFVRFVUSPEEVHFNAOFLBFYJPFPTMFFMFVHBVHFJAENEGVTIGHPWSFU

HPTTMAAGVESGIHJT**PELPEK**JPTIGMPTNJPGJUAUFOXPBFUIEGTIGFJTEIQ

WFXESYIUJTIGIOVEOVIPPOGCWBKTJPGIKMIQWFXESNOOIHFOIHJTCGIXC

SBNRFCDZFEFRLZKNUGRFUTFFIOJITKNRWISAFPTTIQUHJIUYATUUSTOVP

DFFBZPOOGOGVHFIRJOAOFSUTAOIEGGAUWRFUWIKCIYESGATUODKAUG

DXKTIVHFVWPERJOETYHJEHJJAWGAMTEBFYSGCPTDFFSUKLMVHFPAUW

RFQFUJEDCSFCNEVHFGXBNTFFSUCTJQNPHHJUCMKEOVGBXEJVADJASC

CUGRPHIUUOXPIOFEFFAQCRUHRPOTIGNBVUSGOGVHFKNWGSUKGBVIP

P**WIKCIO**YGTIFPDICDPPHBPDUJESGWBUSPOEUJIOIIOJITOATVESNYHTATR

OGCSJVUBVIPPAOFHJUKFGNJPCJUIWGRFCSPPIOI**WIKCIO**AEGIUCPMGAT

WRFVONGTPUTVFYIKSTASUGMPHWPTKBPDUQFPNLPYTIGQVKCLUUCVL

FOEUJOEUBZYHJEHIGDJUEOVAOILFFTIGMPUTJPEYVRJEACNENASUGRJ

The importance of randomised encryption

- It is a problem if the same plaintext block is encrypted to the same ciphertext block every time. This would allow patterns to be found in a long ciphertext.
- We *randomised* encryption schemes to prevent this.
- Typically this is achieved using an *initialisation vector*, IV, which propagates through the entire ciphertext. The IV may need to be:
  - unique;
  - and/ or random.
- Another way to vary the encryption is to include a variable state which is updated with each block.

Efficiency

There are a number of important features of different modes which do not impact security but are really important for practical usage.

- Some modes allow *parallel processing*:
  - sometimes multiple plaintext blocks can be encrypted in parallel;
  - sometimes multiple ciphertext blocks can be decrypted in parallel.
- Some modes result in *error propagation*: a bit error which occurs in the ciphertext results in multiple bit errors in the plaintext after decryption.

Padding

- Some modes, including ECB and CBC, require the plaintext to consist of one or more complete blocks.
- NIST Special Publication 800-38A suggests a padding method as follows:
  1. append a single ‘1’ bit to the data string
  2. pad the resulting string by as few ‘0’ bits, possibly none, as are necessary to complete the final block.
- The padding bits can be removed unambiguously, if the receiver knows that this padding method is used:
  1. remove all trailing ‘0’ bits after the last ‘1’ bit
  2. remove a single ‘1’ bit.
- An alternative to padding is *ciphertext stealing* (see exercises).

Notation overview

- The message is n blocks in length
- P represents the plaintext message
- C represents the ciphertext message
- P_t represents plaintext block t where $1 \leq t \leq n$
- C_t represents ciphertext block t where $1 \leq t \leq n$
- K represent the key
- IV represents the initialisation vector

All modes can be applied to any block cipher. A case of special interest is AES when blocks are 128 bits in length.

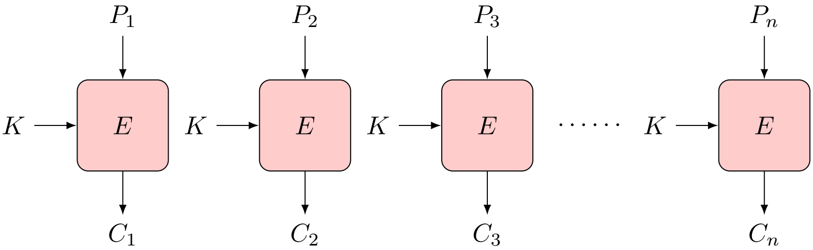
NIST Standards

- Four modes ECB, CBC, CFB and OFB were originally standardised for use with DES in 1980. CTR mode was added in 2001, initially for use with AES.
- SP 800-38A (2001) Confidentiality Modes: ECB, CBC, CFB and OFB. An addendum defines Ciphertext Stealing
- SP 800-38B (2016) CMAC Mode for Authentication
- SP 800-38C (2004, updated 2007) CCM Mode
- SP 800-38D (2007) Galois/Counter Mode (GCM)
- SP 800-38E (2010) XTS-AES Mode for Storage Devices
- SP 800-38F (2012) Key Wrapping
- SP 800-38G (2016) Format-Preserving Encryption

Electronic Code Book (ECB) mode

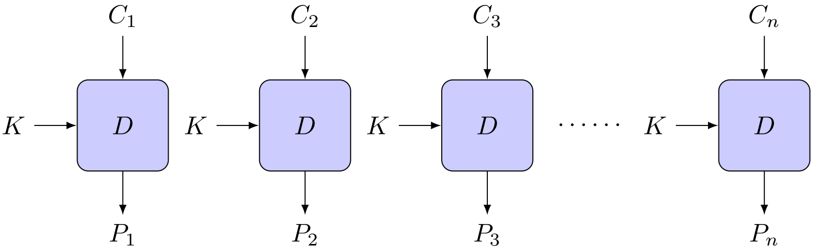
- ECB is the basic mode of a block cipher
- *Encryption*:
  - $C_t = E(P_t, K)$
  - Plaintext block P_t is encrypted with the key K to produce ciphertext block C_t
- *Decryption*:
  - $P_t = D(C_t, K)$
  - Ciphertext block C_t is decrypted with the key K to produce plaintext block P_t

ECB mode encryption



- Blocks $C_1, C_2, \ldots C_n$ are sent

ECB mode decryption



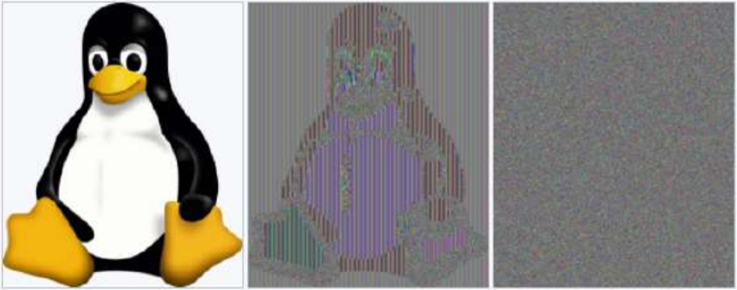
- Blocks $C_1, C_2, \ldots C_n$ are received

ECB mode properties

Randomised	✘
Padding	Required
Error propagation	Errors propagate within blocks
IV	None
Parallel encryption?	✓
Parallel decryption?	✓

- Because it is deterministic, ECB mode is not normally used for bulk encryption.
- If a block is repeated in the plaintext, it will be repeated in the ciphertext!
- Encrypting with ECB mode may reveal *patterns* in the plaintext.

ECB mode – weakness



Original image

Using ECB allows patterns to be easily discerned

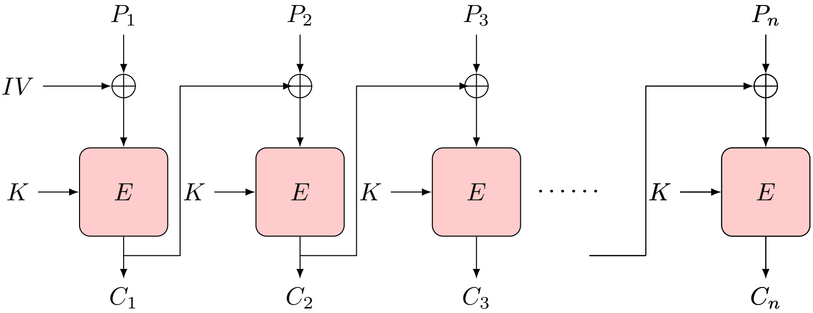
Modes other than ECB result in pseudo-randomness

Source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Cipher Block Chaining (CBC) mode

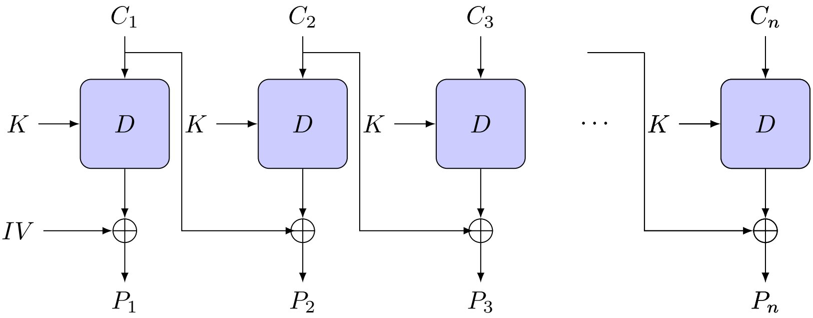
- CBC “chains” the blocks together
- A random initialisation vector *IV* is chosen and sent together with the ciphertext blocks
- *Encryption*:
  - $C_t = E(P_t \oplus C_{t-1}, K)$, where $C_0 = IV$
  - P_t is XOR’d with the previous ciphertext block C_{t-1}, and encrypted with key K to produce ciphertext block C_t
  - IV is used for the value C_0 and sent with $C_1, \ldots C_n$
- *Decryption*:
  - $P_t = D(C_t, K) \oplus C_{t-1}$, where $C_0 = IV$
  - C_t is decrypted with the key K, and XOR’d with the previous ciphertext block C_{t-1} to produce plaintext block P_t
  - As in encryption, C_0 is used as the IV

CBC mode encryption



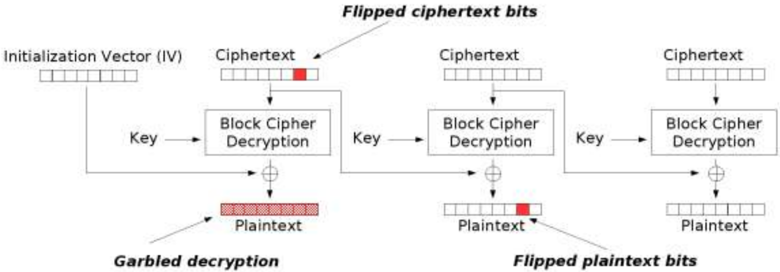
- *IV* and blocks $C_1, C_2, \ldots C_n$ are sent

CBC mode decryption



- IV and blocks $C_1, C_2, \ldots C_n$ are received

CBC mode error propagation



Modification attack or transmission error for CBC

Public domain figure from:

http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

CBC mode properties

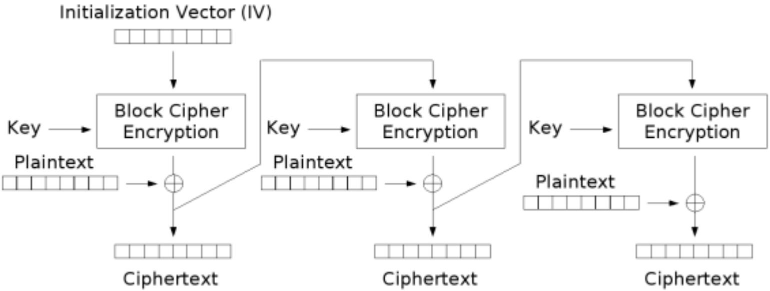
Randomised	✓
Padding	Required
Error propagation	Errors propagate within blocks and into specific bits of next block
IV	Must be random
Parallel encryption?	✘
Parallel decryption?	✓

- Commonly used for bulk encryption
- Common choice for channel protection in all versions of TLS up to TLS 1.2

CFB mode

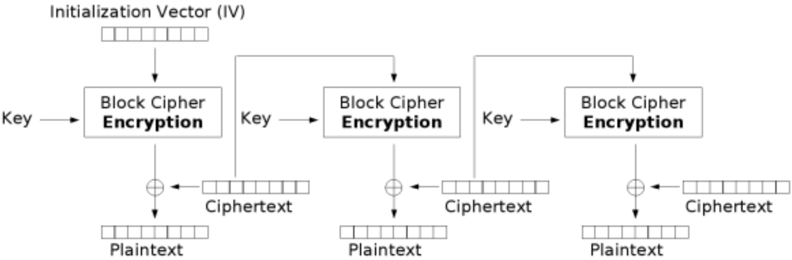
- CFB “feeds” the ciphertext block back into the enciphering/deciphering process, thus “chaining” the blocks together.
- *Encryption*:
  - $C_t = E(C_{t-1}, K) \oplus P_t$, where $C_0 = IV$
- *Decryption*:
  - $P_t = E(C_{t-1}, K) \oplus C_t$
- *Propagation of channel errors*
  - a one-bit change in C_t produces a one-bit change in P_t, and complete corruption of P_{t+1}

CFB mode encryption



Cipher Feedback (CFB) mode encryption

CFB mode decryption



Cipher Feedback (CFB) mode decryption

Self synchronisation in CFB mode

- CFB is a *self-synchronising stream cipher*:
  - Keystream depends on previous ciphertexts, which allows CFB mode to self-synchronise after processing a correct ciphertext block.
  - Assume block C_t is lost in transmission, producing a loss in synchronicity between sender and receiver.
  - Receiver decrypts next received block C_{t+1} as $E(C_{t-1}, K) \oplus C_{t+1}$, which is incorrect (i.e. different from P_t).
  - For the next received block C_{t+2} the receiver computes $E(C_{t+1}, K) \oplus C_{t+2}$, which is the correct plaintext block P_{t+2}. The cipher is back in sync (after losing P_t and P_{t+1}).
- CFB mode can also be defined with a sub-block feedback. In this case re-synchronisation can occur after loss of a sub-block.

CFB mode properties

Randomised	✓
Padding	Not required
Error propagation	Errors occur in specific bits of current block and propagate into next block
IV	Must be random
Parallel encryption?	✘
Parallel decryption?	✓

CFB mode is commonly used when self-synchronisation is useful.

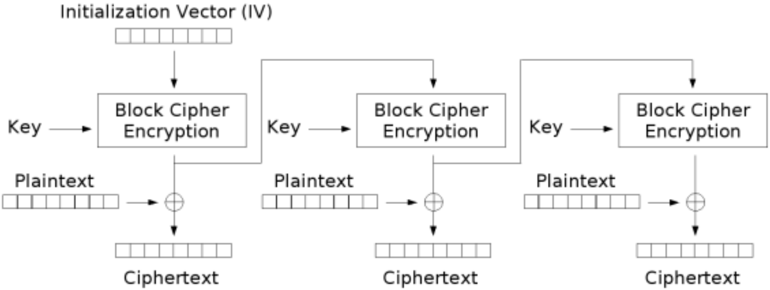
OFB mode

- OFB "feeds" the output block back into enciphering/deciphering process.
- OFB is, in effect, a *synchronous stream cipher*. The keystream is:

$$O_t = E(O_{t-1}, K),$$

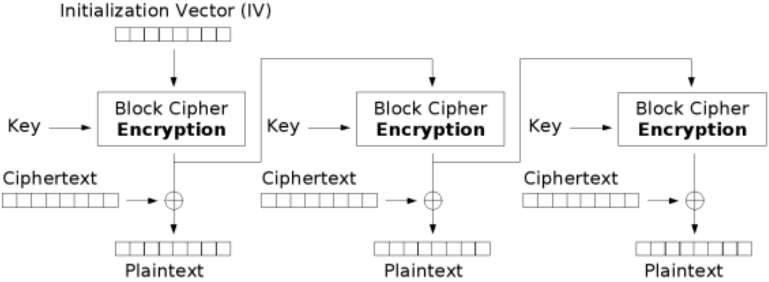
  where $O_0 = IV$ is chosen at random.
- *Encryption*:
    - $C_t = O_t \oplus P_t$
- *Decryption*:
    - $P_t = O_t \oplus C_t$
- *Propagation of channel errors:*
    - a one-bit change in the ciphertext produces a one-bit change in the plaintext at the same location

OFB mode encryption



Output Feedback (OFB) mode encryption

OFB mode decryption



Output Feedback (OFB) mode decryption

OFB mode properties

Randomised	✓
Padding	Not required
Error propagation	Errors occur in specific bits of current block
IV	Must be unique
Parallel encryption?	✘ (but keystream can be computed in advance)
Parallel decryption?	✘

OFB mode is a synchronous stream cipher mode.

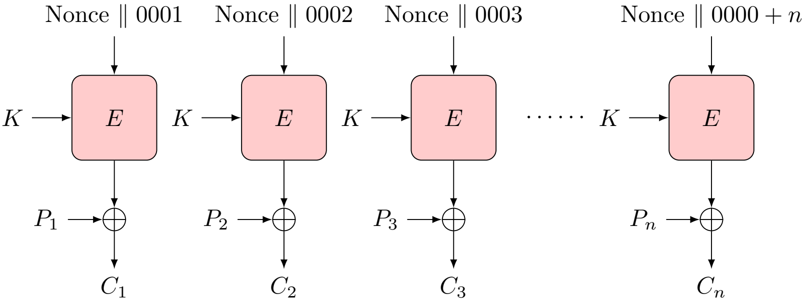
Counter (CTR) mode

- CTR is a *synchronous stream cipher*. The keystream is generated by encrypting successive values of a "counter", initialised using a nonce (randomly chosen value) N:

  $$O_t = E(T_t, K),$$

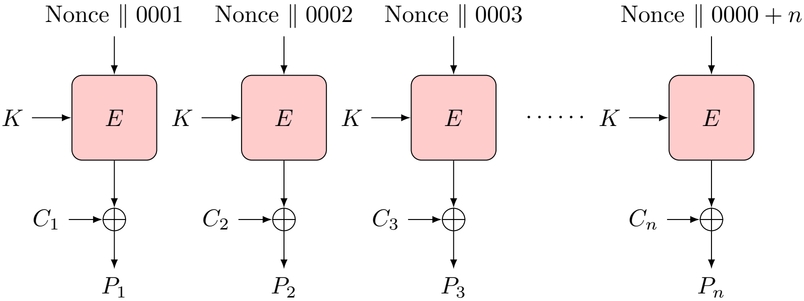
  where $T_t = N \| t$ is the concatenation of the nonce and block number t.
- *Encryption*:
  - $C_t = O_t \oplus P_t$
- *Decryption*:
  - $P_t = O_t \oplus C_t$
- *Propagation of channel errors:*
  - a one-bit change in the ciphertext produces a one-bit change in the plaintext at the same location

CTR mode encryption


$\text{Nonce} \parallel 0001$
$\text{Nonce} \parallel 0002$
$\text{Nonce} \parallel 0003$
$\text{Nonce} \parallel 0000 + n$
K
E
K
E
K
E
$\cdots\cdots$
K
E
P_1
P_2
P_3
P_n
C_1
C_2
C_3
C_n

- Nonce and blocks $C_1, C_2, \ldots C_n$ are sent

CTR mode decryption



- Nonce and blocks $C_1, C_2, \ldots C_n$ are received

CTR mode properties

Randomised	✓
Padding	Not required
Error propagation	Errors occur in specific bits of current block
IV	Nonce must be unique
Parallel encryption?	✓
Parallel decryption?	✓

- A *synchronous stream cipher mode*
- Good for access to specific plaintext blocks without decrypting the whole stream
- Basis for authenticated encryption in TLS 1.2 and TLS 1.3

Principles of (pseudo)random number generation

- Random numbers play a crucial role in cryptography – without strong randomness, we do not have cryptography
  - Think of e.g. encryption keys
- Usually we define some statistical notion. In particular, the two following criteria are important.
  - **Uniform distribution:** I.e. the frequency of ones and zeroes should be approximately equal (for binary outputs).
  - **Idependence:** No output should be predictable given previous/ future outputs.

Randomness in cryptography

- Truly random numbers are used in some applications, but they have drawbacks, such as inefficiency.
- Thus, it is more common to generate sequences of numbers that *appear to be random* but are not random
- In particular, we must ensure that *an adversary cannot predict* future elements
- Cryptographic applications typicaly make use of algorithmic techinques for random number generation.
- These algorithms are *deterministic*, and therefore produce sequences that are not statistically random.
- We refer to these numbers as *pseudorandom* numbers.

Randomness

- Any specific string of bits (number) is exactly as random as any other string – which is why we look at *distributions* (in a statistics sense)
- We think instead in terms of *generators* of random strings
  - A *true random number generator* (TRNG) is a physical process which outputs each valid string independently with equal probability
  - A *pseudo random number generator* (PRNG) is a deterministic algorithm which approximates a TRNG
- We may use a TRNG to provide a *seed* for a PRNG

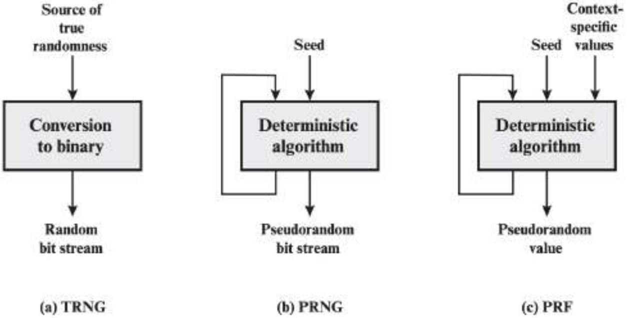
TRNGs

- NIST Special Publication SP 800-90B (2018) provides a framework for design and validation of TRNG algorithms called *entropy sources*.
- The entropy source includes a physical noise source, a digitization process and post-processing stages. The output is any requested number of bits.
- The standard specifies many statistical tests for validating the suitability of entropy sources
- An important additional requirement is a periodic *health test* to ensure continuing reliable operation
- Intel introduced TRNGs into Ivy Bridge processors in 2012

PRNGs

- NIST Special Publication SP 800-90A (June 2015) recommends specific PRNG algorithms named *Deterministic Random Bit Generators* (DRBG) based on:
  - hash functions (we look at these in a later lecture);
  - a specific MAC known as HMAC (also in a later lecture);
  - block ciphers in counter mode.
- Each generator takes a seed as input and outputs a bit string before updating its state
- The seed should be updated after some number of calls
- The seed can be obtained from a TRNG

TRNGs, PRNGs and PRFs



TRNG = true random number generator
PRNG = pseudorandom number generator
PRF = pseudorandom function

Figure 8.1 Random and Pseudorandom Number Generators

Cryptography and Network Security, Stallings

TRNGs input into PRNGs

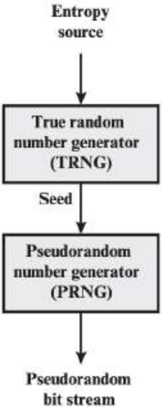

Entropy source
True random number generator (TRNG)
Seed
Pseudorandom number generator (PRNG)
Pseudorandom bit stream

Figure 8.2 Generation of Seed Input to PRNG

Cryptography and Network Security, Stallings

Functions of DRBGs

The SP 800-90A standard defines a general model for DRBGs with some functions

Instantiate This sets the initial state of the DRBG using a seed

Generate This provides an output bit string for each request

Reseed Inputs a new random seed and updates the state

Test Checks correct operation of the other functions

Uninstantiate Deletes (“zeroises”) the state of the DRBG

Security of DRBGs

The standard defines the security of DRBGs in terms of the ability of an attacker to *distinguish reliably* between its output and a truly random string. Two properties are defined.

Backtracking resistance An attacker who obtains the current state of the DRBG should not be able to distinguish between the output of earlier calls to the DRBG generate function and random strings.

Forward prediction resistance An attacker who obtains the current state of the DRBG should not be able to distinguish between the output of later calls to the DRBG generate function and random strings.

CTR_DRBG

- Uses a block cipher in CTR mode. AES with 128-bit keys is one recommended option.
- The DBRG is initialised with a seed whose length is equal to the key length plus the block length, so $128 + 128 = 256$ bits for AES with 128-bit keys.
- From a high entropy seed a key K and state (counter) value V are derived. There is no separate nonce as in normal CTR mode.
- Counter mode encryption is then run iteratively (with no plaintext added) and the output blocks form the output.

Update function in CTR_DRBG

- The Update function is used in the Initialise, Generate and Reseed functions to **generate new key and state**
- Inputs are current key K and state (counter) V and optional data input D
- Output is a new key K' and state (counter) V'
- When block and key size are the same, computation is:
  - Generate new block $O_1 = E(V, K)$
  - Increment V
  - Generate new block $O_2 = E(V, K)$
  - $K' \parallel V' = (O_1 \parallel O_2) \oplus D$
- Updating provides backtracking resistance

Instantiate, generate and reseed in CTR_DRBG

Instantiate calls the Update function with *D* equal to high entropy seed, and *K* and *V* all zero strings

Generate computes up to 2^{19} bits by running CTR mode output from the current state. Then the Update function is called with *D* empty

Reseed calls the Update function with *D* equal to high entropy input, and *K* and *V* in the current state.

- The standard restricts the output to 2^{48} requests to the Generate function before Reseed must be called
- Each Reseed call provides forward prediction resistance and backtracking resistance

Dual_EC_DRBG

- Older standard (December 2012) includes Dual_EC_DRBG
- Based on elliptic curve discrete logarithm problem (we look at this in a later lecture)
- Much slower than other DRBGs in the standard
- Based on hard problem but no security proof exists. In fact NTNU's Kristian Gjøsteen and others showed in 2006 that the output has an observable bias
- In December 2013 the press reported a secret ten million dollar deal between the NSA and RSA Security company to use Dual_EC_DRBG as the default PRNG in its software suite

http://blog.cryptographyengineering.com/2013/09/the-many-flaws-of-dualecdrbg.html

Cloudfare

- Cloudfare is a company located in San Francisco, USA
- They provide:
  - Content delivery network services
  - Cloud cybersecurity
  - DDoS mitigation
- Cloudflare is used by more than 20 percent of the Internet for its web security services as of 2022.

https://en.wikipedia.org/wiki/Cloudflare
https://www.cloudflare.com/en-gb/

How to generate randomness?

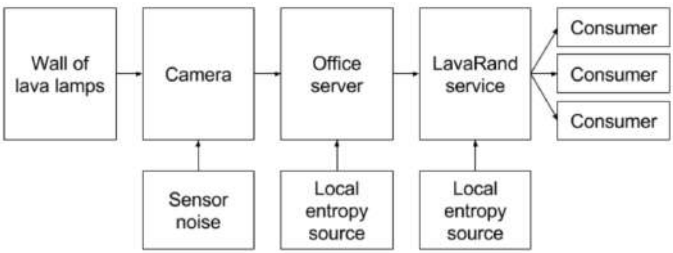
- Cryptographically-secure pseudorandom number generators (CSPRNGs) are algorithms which, provided an input which is itself unpredictable, produce a much larger stream of output which is also unpredictable
- But only half of the equation – they need an unpredictable input
- High-accuracy measurements are unpredictable! e.g. temperature

LavaRand

https://blog.cloudflare.com/randomness-101-lavarand-in-production

LavaRand

The pipeline – LavaRand is only a backup



https://blog.cloudflare.com/lavarand-in-production-the-nitty-gritty-technical-details

Lecture 7: Hash functions, MACs and authenticated encryption

TTM4135

Relates to Stallings Chapters 11 and 12

Spring Semester, 2025

Motivation

- Hash functions are versatile cryptographic functions used as a building block for authentication
- Message authentication codes (MACs) are a symmetric key cryptographic mechanisms providing authentication and integrity services
- The standardised MAC, HMAC, can be based on many different hash functions and is often used in the TLS protocol
- Authenticated encryption combines confidentiality and authentication in one mechanism
- GCM is a standardised authenticated encryption algorithm also often used in TLS

Outline

- Hash functions
  - Security properties
  - Iterated hash functions
  - Standardized hash functions
  - Using hash functions
- Message Authentication Codes (MACs)
  - MACs from hash functions – HMAC
- Authenticated encryption
  - Combining encryption and MAC
  - Galois Counter Mode (GCM)
- Passwords and hashing

Outline

Hash functions
- Security properties
- Iterated hash functions
- Standardized hash functions
- Using hash functions

Message Authentication Codes (MACs)
- MACs from hash functions – HMAC

Authenticated encryption
- Combining encryption and MAC
- Galois Counter Mode (GCM)

Passwords and hashing

Hash functions

A *hash function* H is a public function such that:

- H is simple and fast to compute
- H takes as input a message M of arbitrary length and outputs a message digest $H(M)$ of fixed length

Outline

- Hash functions
  - Security properties
  - Iterated hash functions
  - Standardized hash functions
  - Using hash functions
- Message Authentication Codes (MACs)
  - MACs from hash functions – HMAC
- Authenticated encryption
  - Combining encryption and MAC
  - Galois Counter Mode (GCM)
- Passwords and hashing

Security properties of hash functions

Collision resistant:

- It should be infeasible to find any two different values x_1 and x_2 with $H(x_1) = H(x_2)$.

One-way (or preimage resistant):

- Given a value y it should be infeasible to find any input x with $H(x) = y$.

Second-preimage resistant:

- Given a value x_1 it should be infeasible to find a different value x_2 with $H(x_1) = H(x_2)$.

An attacker who can break second-preimage resistance can also break collision resistance.

The birthday paradox

- In a group of 23 randomly chosen people, the probability that at least two have the same birthday is over 0.5.
- In general, if we choose around $\sqrt{M}$ values from a set of size M, the probability of getting two values the same is around 0.5
- Suppose a hash function H has an output size of k bits. If H is regarded as a random function then $2^{k/2}$ trials are enough to find a collision with probability around 0.5.
- Today 2^{128} trials would be considered infeasible. Therefore, in order to satisfy collision resistance, hash functions should have output of at least 256 bits.

Birthday paradox example, $M = 100$

# trials	Collision prob.	# trials	Collision prob.
1	0	13	.55727
2	.01000	14	.61483
3	.02980	15	.66876
4	.05891	16	.71845
5	.09656	17	.76350
6	.14174	18	.80371
7	.19324	19	.83905
8	.24972	20	.86964
9	.30975	21	.89572
10	.37188	22	.91762
11	.43470	23	.93575
12	.49689	24	.95053

Outline

Hash functions
- Security properties
- Iterated hash functions
- Standardized hash functions
- Using hash functions

Message Authentication Codes (MACs)
- MACs from hash functions – HMAC

Authenticated encryption
- Combining encryption and MAC
- Galois Counter Mode (GCM)

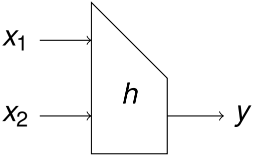
Passwords and hashing

Iterated hash functions

- Cryptographic hash functions need to take arbitrary-sized input and produce a fixed size output.
- As we saw from block ciphers, we can process arbitrary-sized data by having a function that processes fixed-sized data and use it repeatedly.
- An *iterated hash function* splits the input into blocks of fixed size and operates on each block sequentially using the same function with fixed size inputs.
- Merkle–Damgård construction: use a fixed-size *compression function* applied to multiple blocks of the message.

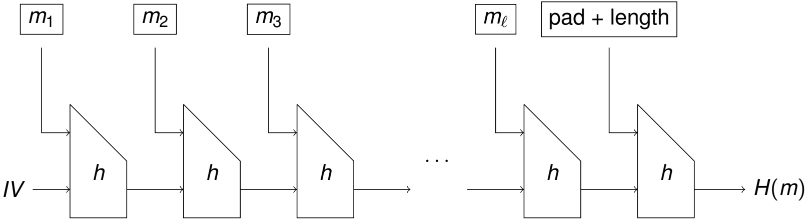
Compression function h

h takes two n-bit input strings x_1 and x_2 and produces an n bit output string y.



Merkle–Damgård construction

1. Break message m into n-bit blocks $m_1 \| m_2 \| \ldots \| m_\ell$.
2. Add padding and an encoding of the length of m. This process may, or may not, add one block.
3. Input each block into compression function h along with chained output; use IV to get started.



Use of Merkle–Damgård construction

Security: If compression function h is collision-resistant, then hash function H is collision-resistant. Proof on blackboard.

But also some security weaknesses:

- Length extension attack: once you have one collision, easy to find more
- Second-preimage attacks not as hard as they should be

Many standard, and former standard, hash functions are Merkle–Damgård constructions: MD5, SHA-1, SHA-2 family

Outline

Hash functions

- Security properties
- Iterated hash functions
- Standardized hash functions
- Using hash functions

Message Authentication Codes (MACs)

- MACs from hash functions – HMAC

Authenticated encryption

- Combining encryption and MAC
- Galois Counter Mode (GCM)

Passwords and hashing

MDx family

- Proposed by Rivest and widely used throughout 1990s
- Deployed family members were MD2, MD4 and MD5
- All have 128-bit output
- All of them are broken (concrete collisions have been found)
- In 2006, MD5 collisions could be found in one minute on a PC

SHA-0 and SHA-1

- Based on MDx family design but larger output and more complex
- Originally Secure Hash Algorithm published by US standard agency NBS (now called NIST) in 1993 and later given name SHA-0
- Replaced by SHA-1 in 1995 with very small change to algorithm
- Both SHA-0 and SHA-1 have 160 bit output.
- SHA-0 has been broken (collisions found in 2004)
- First SHA-1 collision found in 2017 - attack is 100 000 times faster than brute force search

SHA-2 family

Developed in response to (real or theoretical) attacks on MD5 and SHA-1.

	Hash size	Block size	Security match
SHA-224	224 bits	512 bits	2 key 3DES
SHA-512/224	224 bits	1024 bits	2 key 3DES
SHA-256	256 bits	512 bits	AES-128
SHA-512/256	256 bits	1024 bits	AES-128
SHA-384	384 bits	1024 bits	AES-192
SHA-512	512 bits	1024 bits	AES-256

- Standardized in FIPS PUB 180-4 (August 2015)
- Known collectively as SHA-2

Padding in the SHA-2 family

- The message length field is:
  - 64 bits when the block length is 512 bits
  - 128 bits when the block length is 1024 bits
- There is always at least one bit of padding[1]. After the first ‘1’ in the padding, enough ‘0’ bits are added so that after the length field is added there is an exact number of complete blocks.
- Adding the padding and length field sometimes adds an extra block and sometimes does not.

[1] To avoid trivial second preimage attacks.

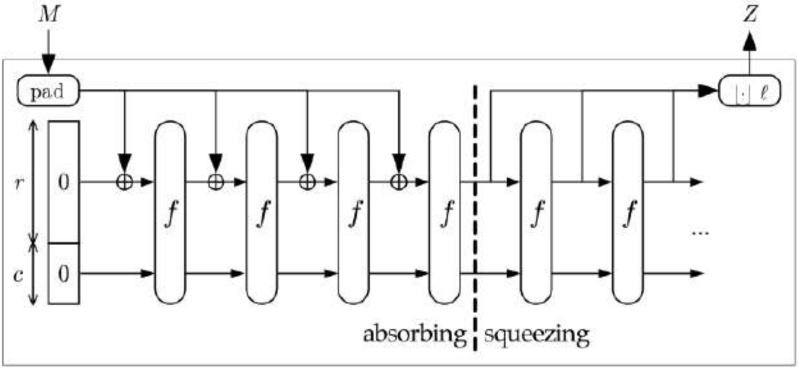
SHA-3

- Late 2000s seen to be a crisis in hash function design
- MDx and SHA family are all based on the same basic design and there have been several unexpected attacks on these in recent years
- In November 2007, NIST announced a competition for a new hash standard called SHA-3
    - Entries closed October 2008; 64 original candidates
    - 14 went through to Round 2, with 5 finalists announced in December 2010
    - Keccak selected as winner in October 2012.
    - Keccak doesn't use compression function as in Merkle–Damgård construction. Instead it uses a *sponge* construction
    - Standardized in FIPS PUB 202, August 2015

The sponge construction

- Input is padded and broken down into blocks of *r* bits
- The *b* bits of the state are initialized to zero, and the sponge function proceeds:
  - Absorbing phase: the input blocks are XOR'ed into the *r* first bits of the state, and the function *f* is applied iteratively
  - Squeezing phase: the first *r* bits of the state are returned as output blocks, interleaved with applying the function *f*.
    - The number of output blocks is chosen by the user.
    - The last *c* bits of the state are never directly affected by the input blocks and are never output during the squeezing phase.
  - Since the input/ output sizes can be arbitrarily long, the sponge construction can be used to build various primitives (hash functions, stream ciphers, MAC etc).
    - Long input, short output → Hash function
    - Short input, large output → Key stream

The Keccak function



Source: https://keccak.team/sponge_duplex.html

Outline

Hash functions

- Security properties
- Iterated hash functions
- Standardized hash functions
- Using hash functions

Message Authentication Codes (MACs)

- MACs from hash functions – HMAC

Authenticated encryption

- Combining encryption and MAC
- Galois Counter Mode (GCM)

Passwords and hashing

Uses of hash functions

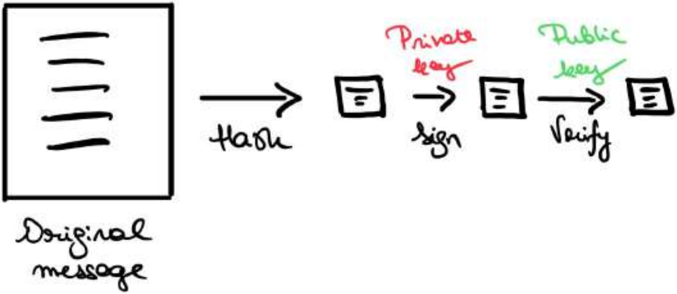
Hash functions have many uses. Note that applying a hash function is *not* encryption:

- Hash computation does not depend on a key*.
- It is not not designed to go backwards to find the input (generally not possible).

While hash functions alone do not provide data authentication, they often help in achieving it:

- Authenticate the hash of a message to authenticate the message.
- Building block for Message Authentication Codes (see HMAC below).
- Building block for signatures (later lecture).

Reminder: signatures



Hash functions and keys

- Sometimes we write hash functions as taking a key *s* as an input
- $H^s(x) = H(s, x)$
- It must be hard to find a collision for a randomly generated key *s*
- The key is *not* kept secret; collision resistance must hold even if the adversary has the key *s*
- This is why we[2] write H^s and not H_s

[2] Katz-Lindell book

Storing passwords for login

Usual to store user passwords on servers using hash functions

- Store salted hashes of passwords: pick random *salt*, compute $h = H(pw, salt)$, store $(salt, h)$
    - easy to check entered password pw': compare stored h and computed $H(pw', salt)$
    - hard to recover pw from $h = H(pw)$ assuming H is preimage resistant
    - attacker needs to store a different dictionary for each *salt*

Note that using a *slower* hash function slows down password guessing

Outline

- Hash functions
  - Security properties
  - Iterated hash functions
  - Standardized hash functions
  - Using hash functions
- Message Authentication Codes (MACs)
  - MACs from hash functions – HMAC
- Authenticated encryption
  - Combining encryption and MAC
  - Galois Counter Mode (GCM)
- Passwords and hashing

Message Authentication Codes (MAC)s

- Recall one of the goals of cryptography is to enable *secure communications*
  - But what does this mean?
- We have covered *secrecy* so far (i.e. *hiding* the message)
- But *integrity can be even more important*
- For example, a bank receives a transfer request from user Alice to user Bob in the amount of $ 10,000
  - Did it really come from Alice?
  - Is the amount correct? Was it modified during transmission?

Encryption vs Message Authentication

- Encryption *does not* guarantee message integrity
- These are *different* notions
- Recall that we saw that flipping certain bits in the ciphertext results in flipping certain bits in the plaintext
- If the adversary also has partial information about the plaintext (e.g. an estimate of the amount that is being sent), it can predict with some accuracy what the changes are
    - Even the OTP is vulnerable to this, so this does not mean that the encryption scheme is not secure!
- An adversary can also randomly change the ciphertext, to ruin the underlying message!

Message Authentication Code (MAC)s

- A *message authentication code (MAC)* is a cryptographic mechanism used for message integrity and authentication
- On input a secret key K and an arbitrary length message M, a MAC algorithm outputs a fixed-length tag:

$$T = \mathrm{MAC}(M, K),$$

- A MAC is a symmetric key algorithm: sender and receiver both have the secret key K
- The sender sends the pair (M, T) but M may or may not be encrypted
- The recipient recomputes the tag $T' = \mathrm{MAC}(M', K)$ on the received message M' and checks that $T' = T$

MAC properties

The basic security property of a MAC is called *unforgeability*:

- It is not feasible to produce a message M and a tag T such that $T = \mathrm{MAC}(M, K)$ without knowledge of the key K

The more complete notion of security is *unforgeability under chosen message attack*:

- The attacker is given access to a *forging oracle*: on input any message M of the attacker's choice the MAC tag $T = \mathrm{MAC}(M, K)$ is returned
- It is not feasible for the attacker to produce a valid (M, T) pair that was not already asked to the oracle

Not a signature scheme, but can be thought of as the symmetric version of a signature scheme. Here the point is that only *authorised* entities can authenticate messages.

MAC properties

The basic security property of a MAC is called *unforgeability*:

- It is not feasible to produce a message M and a tag T such that $T = \mathrm{MAC}(M, K)$ without knowledge of the key K

The more complete notion of security is *unforgeability under chosen message attack*:

- The attacker is given access to a *forging oracle*: on input any message M of the attacker's choice the MAC tag $T = \mathrm{MAC}(M, K)$ is returned
- It is not feasible for the attacker to produce a valid (M, T) pair that was not already asked to the oracle

Not a signature scheme, but can be thought of as the symmetric version of a signature scheme. Here the point is that only *authorised* entities can authenticate messages.

Outline

- Hash functions
  - Security properties
  - Iterated hash functions
  - Standardized hash functions
  - Using hash functions
- Message Authentication Codes (MACs)
  - MACs from hash functions – HMAC
- Authenticated encryption
  - Combining encryption and MAC
  - Galois Counter Mode (GCM)
- Passwords and hashing

HMAC

- Proposed by Bellare, Canetti, Krawczyk in 1996
- Built from any iterated cryptographic hash function H, e.g., MD5, SHA-1, SHA-256, …
- Standardized and used in many applications including TLS and IPsec
- Details in FIPS-PUB 198-1 (July 2008)

HMAC construction

Let H be any iterated cryptographic hash function. Then define:

$$\mathrm{HMAC}(M, K) = H(\ (K \oplus \mathrm{opad}) \parallel H((K \oplus \mathrm{ipad}) \parallel M)\ )$$

where

- M: message to be authenticated
- K: key padded with zeros to be the block size of H
- opad: fixed string `0x5c5c5c...5c`
- ipad: fixed string `0x363636...36`
- $\parallel$ denotes concatenation of bit strings.
- HMAC is secure (unforgeable) if H is collision resistant or if H is a pseudorandom function.

HMAC

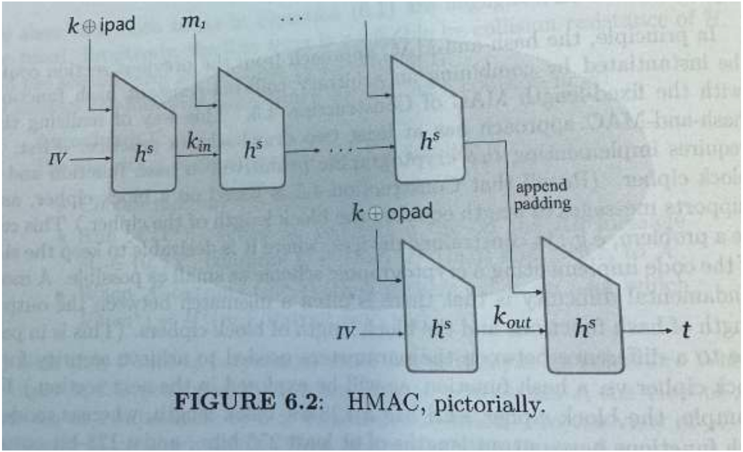


FIGURE 6.2: HMAC, pictorially.

Source: Katz-Lindell book, third edition.

Security of HMAC

- *Security:* HMAC is secure if H is collision resistant or if H is a pseudorandom function.
- HMAC is designed to resist length extension attacks (even if H is a Merkle–Damgård hash function).
- HMAC is often used as a *pseudorandom function* for deriving keys in cryptographic protocols.

H is h^s in the previous slide.

Outline

- Hash functions
  - Security properties
  - Iterated hash functions
  - Standardized hash functions
  - Using hash functions
- Message Authentication Codes (MACs)
  - MACs from hash functions – HMAC
- Authenticated encryption
  - Combining encryption and MAC
  - Galois Counter Mode (GCM)
- Passwords and hashing

Outline

- Hash functions
  - Security properties
  - Iterated hash functions
  - Standardized hash functions
  - Using hash functions
- Message Authentication Codes (MACs)
  - MACs from hash functions – HMAC
- Authenticated encryption
  - Combining encryption and MAC
  - Galois Counter Mode (GCM)
- Passwords and hashing

Authenticated encryption

Suppose Alice and Bob have a shared key K.

Suppose Alice has a message M that she wants to send to Bob with *confidentiality* and *authenticity/integrity*.

How should she do this? Two options:

1. Split the key K into two parts (K_1 and K_2), encrypt with K_1 to provide confidentiality and use K_2 with a MAC to provide authenticity and integrity.
2. Use a dedicated algorithm which provides both properties – this is called *authenticated encryption*.

Combining encryption and message authentication

Three possible ways to combine encryption and MAC are:

Encrypt and MAC: encrypt M, apply MAC to M, and send the two results

MAC then encrypt: apply MAC to M to get tag T, then encrypt $M \| T$, and send the ciphertext

Encrypt then MAC: encrypt M to get ciphertext C, then MAC C, and send the two results

It turns out that *encrypt-then-MAC* is the safest approach.

- $C = \mathrm{Enc}(M, K_1)$
- $T = \mathrm{MAC}(C, K_2)$
- send $C \| T$

Encrypt and MAC

- No integrity on the ciphertext! Only on the plaintext, which can be problematic.
- This may not achieve the most basic level of secrecy.
- Even a strongly secure MAC does not guarantee *anything* about secrecy.
- A MAC may leak information about m in the tag t.
    - Think of a MAC who always outputs the first bits of m in the tag.

MAC then Encrypt

- Plaintext integrity only, but this time the MAC is encrypted.
- Because we pad the message with the tag, we have two sources of decryption error:
  - Padding may be incorrect.
  - Tag may not verify.
- An attacker can distinguish between the failures and exlpoit this.
- This type of attack has been carried out against some TLS configurations.

Authenticated encryption with associated data (AEAD)

- An AEAD algorithm is a symmetric key cryptosystem
- Inputs to the AEAD encryption process are:
  - a message *M*
  - associated data *A*
  - the shared key *K*
- The AEAD output *O* may contain different elements such as a ciphertext and tag
- The sender sends *O* and *A* to the recipient
- The receiver either accepts the message *M* and data *A*, or reports *failure*
- Any AEAD algorithm should provide
  - confidentiality for *M*
  - authentication for both *M* and *A*

Outline

Hash functions
- Security properties
- Iterated hash functions
- Standardized hash functions
- Using hash functions

Message Authentication Codes (MACs)
- MACs from hash functions – HMAC

Authenticated encryption
- Combining encryption and MAC
- Galois Counter Mode (GCM)

Passwords and hashing

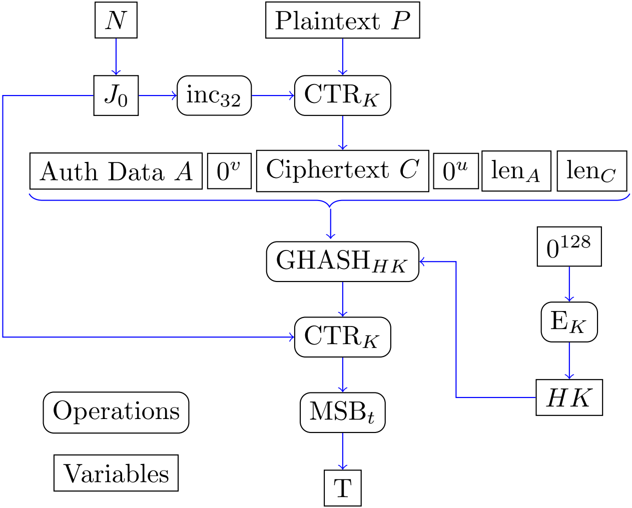
Galois Counter Mode (GCM)

- A block cipher mode providing AEAD
- Most commonly used mode in web-based TLS
- Combines CTR mode on a block cipher (typically AES) with a special keyed hash function called GHASH.
- Standard definition in NIST SP-800 38D
- Due to hardware support of AES and carry-less addition in modern Intel chips, GCM using AES can be faster than using AES with HMAC.

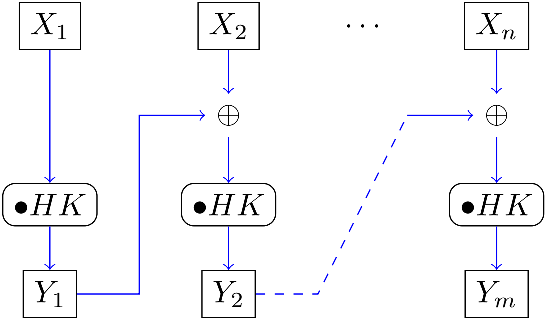
GCM algorithm

- GHASH uses multiplication in the finite field $GF(2^{128})$
    - Generated by the polynomial $x^{128} + x^7 + x^2 + 1$.
- Inputs are plaintext P, authenticated data A, and nonce N
- Values u and v are the minimum number of 0s required to expand A and C to complete blocks
- Outputs are ciphertext C and tag T. The length of A, len_A, and the length of C, len_C, are 64-bit values
- In TLS the length of T is $t = 128$ bits and the nonce N is 96 bits. The initial block input to CTR mode of E (denoted CTR in diagram) is $J_0 = N \parallel 0^{31} \parallel 1$.
- The function inc_{32} increments the right-most 32 bits of the input string by 1 modulo 2^{32}

GCM algorithm



GHASH



- Output is $Y_m = GHASH_{HK}(X_1, \ldots, X_m)$
- Operation $\bullet$ is multiplication in the finite field $GF(2^{128})$
- $HK = E(0^{128}, K)$ is the hash subkey.

GCM decryption

- The elements transmitted to the receiver are the ciphertext C, the nonce N, the tag T and the authenticated data A.
- All elements required to recompute the tag T are available to the receiver who shares key K. The tag is recomputed and checked with received tag. If tags do not match then output is declared invalid.
- If the tag is correct then the plaintext can be recomputed by generating the same key stream, from CTR mode, as is used for encryption.

Outline

Hash functions
- Security properties
- Iterated hash functions
- Standardized hash functions
- Using hash functions

Message Authentication Codes (MACs)
- MACs from hash functions – HMAC

Authenticated encryption
- Combining encryption and MAC
- Galois Counter Mode (GCM)

Passwords and hashing

Cryptography and passwords

Cryptography needs high-entropy secrets:

- RC4, AES-128: 128 bit secret key
- HMAC-SHA1: 160-bit secret key
- AES-256, HMAC-SHA256: 256-bit secret key

128 bits = about 23 character alphanumeric secret

Humans can only memorize low entropy passwords:

- RockYou.com password database compromised in 2009; *password entropy 21.1 bits*

Uses of passwords

Applications of passwords:

- *Login*: system stores password to compare with the value the user types at login to decide whether to allow access. Obviously don’t want to store passwords in plaintext on disk.
- *Key derivation*: user remembers a password that will be used to derive a key for encryption, e.g., disk encryption.

Dictionary attacks against passwords

- Attacker obtains a dictionary of passwords sorted by approximate frequency of use.
- Attacker iterates through dictionary from most frequent to least frequent passwords.

Storing passwords for login

How can we store user passwords on hard disks for checking at login?

- Store passwords in plaintext: Bad idea; anyone who gets hard disk can learn password.
- Store passwords encrypted using a secret key: Where do you store the secret key? Becomes a chicken-and-egg problem.
- Store hashes of passwords: $h = H(pw)$
    - Pro: easy to check entered password pw': compare stored h and computed $H(pw')$
    - Pro: hard to recover pw from $h = H(pw)$ assuming H is preimage resistant
    - Con: attacker could store a dictionary of $pw, H(pw)$ and compare with stolen h

Storing passwords for login

How can we store user passwords on hard disks for checking at login?

- Store salted hashes of passwords: pick random *salt*, compute $h = H(pw, salt)$, store $(salt, h)$
  - Pro: easy to check entered password pw': compare stored h and computed $H(pw', salt)$
  - Pro: hard to recover pw from $h = H(pw)$ assuming H is preimage resistant
  - Pro: attacker needs to store a different dictionary for each *salt*
  - Con: doesn't slow down per-password attacks

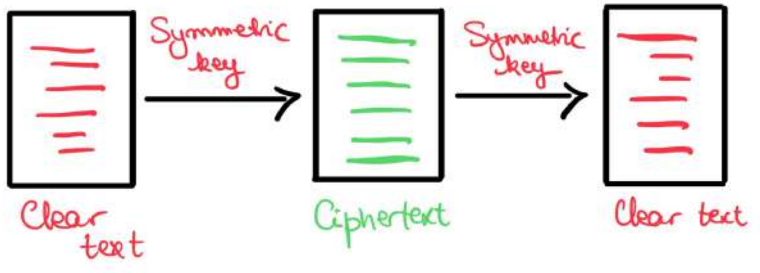
Lecture 8: Number Theory for Public Key Cryptography

TTM4135

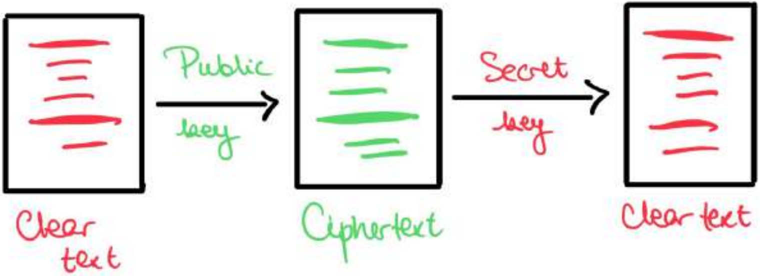
Relates to Stallings Chapters 2 and 5

Spring Semester, 2025

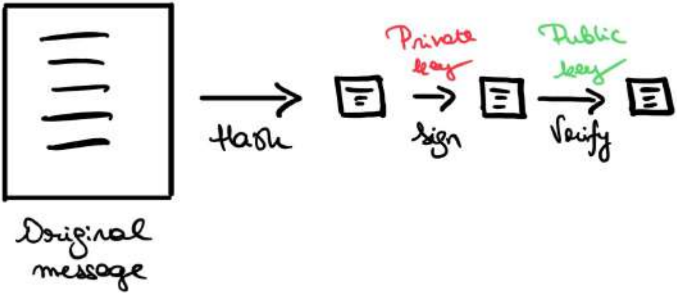
Reminder – symmetric key encryption



Reminder – public key encryption



Reminder – signatures



Motivation

- Number theoretic problems are at the foundation of public key cryptography (e.g encryption, signatures) in use today.
- In order to use these problems we need efficient ways to generate large prime numbers.
- We also need to define hard computational problems which we can base our cryptosystems on.

Outline

Chinese remainder theorem

Euler function ϕ

Testing for primality
- Fermat Test
- Miller–Rabin Test

Some Basic Complexity Theory

Factorisation problem

Discrete logarithm problem

Chinese remainder theorem

Theorem

Let $d_1, d_2, \ldots, d_r$ be pairwise relatively prime and $n = d_1 d_2 \ldots d_r$. Given any integers c_i there exists a unique integer x with $0 \leq x < n$ such that

$$\begin{aligned} x &\equiv c_1 \pmod{d_1} \\ x &\equiv c_2 \pmod{d_2} \\ &\vdots \\ x &\equiv c_r \pmod{d_r} \end{aligned}$$

In fact $x \equiv \sum (\frac{n}{d_i})\, y_i\, c_i \pmod{n}$ where $y_i \equiv (\frac{n}{d_i})^{-1} \pmod{d_i}$.

Example

$$\text{Solve} \quad \begin{aligned} x &\equiv 5 \pmod{6} \\ x &\equiv 33 \pmod{35} \end{aligned}$$

Since 6 and 35 are relatively prime we can use CRT. Set $n = 6 \times 35 = 210$.

$$\begin{aligned} \tfrac{210}{6} y_1 &\equiv 1 \pmod{6} & \qquad \tfrac{210}{35} y_2 &\equiv 1 \pmod{35} \\ 35 y_1 &\equiv 1 \pmod{6} & 6 y_2 &\equiv 1 \pmod{35} \\ y_1 &\equiv 5 \pmod{6} & y_2 &\equiv 6 \pmod{35} \end{aligned}$$

$$\begin{aligned} x &\equiv \sum \left(\frac{n}{d_i}\right) y_i\, c_i \pmod{n} \\ &\equiv (35 \times 5 \times 5) + (6 \times 6 \times 33) \pmod{210} \\ &\equiv 175 \times 5 + 36 \times 33 \pmod{210} \\ &\equiv 173 \pmod{210} \end{aligned}$$

Euler function ϕ

Definition

For a positive integer *n*, the Euler function $\phi(n)$ denotes the number of positive integers less than *n* and relatively prime to *n*.

- Recall that *a* and *b* relatively prime is the same as $\gcd(a, b) = 1$.
- The set of positive integers less than *n* and relatively prime to *n* form the reduced residue class $\mathbb{Z}_n^*$.
  - So in particular, $\phi(n)$ gives us the *size* of $\mathbb{Z}_n^*$.

Example

$\phi(10) = 4$ since 1,3,7,9 are each relatively prime to 10.

$\mathbb{Z}_{10}^* = \{1, 3, 7, 9\}$

Properties of $\phi(n)$

1. $\phi(p) = p - 1$ for p prime
2. $\phi(pq) = (p-1)(q-1)$ for p and q distinct primes
3. Let $n = p_1^{e_1} \dots p_t^{e_t}$ where p_i are distinct primes. Then

$$\phi(n) = \prod_{i=1}^{t} p_i^{e_i - 1}(p_i - 1)$$

where $\prod$ represents the product

Example

$$\begin{array}{rcccl} \phi(15) & = & 2 \times 4 & = & 8 \\ \phi(24) & = & 2^2(2-1)3^0(3-1) & = & 8 \\ & & (\text{where } 24 = 2^3 \times 3) & & \end{array}$$

Two important theorems

Theorem (Fermat)

Let p be a prime. Then

$$a^{p-1} \bmod p = 1$$

for all integers a with $1 < a < p - 1$

Theorem (Euler)

$$a^{\phi(n)} \bmod n = 1$$

if $\gcd(a, n) = 1$*.*

- When p is prime then $\phi(p) = p - 1$ so Fermat's theorem is a special case of Euler's theorem

Testing for primality

- Testing for primality by trial division is not practical except for very small numbers
- There are a number of fast methods which are *probabilistic*: they require random input and can fail in exceptional circumstances
- In 2002, three Indian mathematicians, Agrawal, Saxena and Kayal, found a polynomial time deterministic primality test. Although a huge theoretical breakthrough, the probabilistic methods are still used in practice.
- We examine one of the simplest tests: the *Fermat primality test* and then extend it to the *Miller–Rabin test*

Outline

Chinese remainder theorem

Euler function ϕ

Testing for primality
- Fermat Test
- Miller–Rabin Test

Some Basic Complexity Theory

Factorisation problem

Discrete logarithm problem

Fermat primality test

- Recall that Fermat's theorem says that *if* a number p is prime then $a^{p-1} \bmod p = 1$ for all a with $\gcd(a, p) = 1$
- If we examine a number n and find that $a^{n-1} \bmod n \neq 1$ then we know that n is *not* prime
- This is essentially the Fermat primality test: if a number satisfies Fermat's equation then we assume that it is prime
- The Fermat primality test can fail with some probability
- We reduce the failure probability by repeating the test with different base values a

Fermat primality test

Inputs:
- n: a value to test for primality;
- k: a parameter that determines the number of times to test for primality

Output: `composite` if n is composite, otherwise `probable prime`

Algorithm:

```
repeat $k$ times:
    1. pick $a$ randomly in the range $1 < a < n - 1$
    2. if $a^{n-1} \bmod n \neq 1$ then return composite
return probable prime
```

Effectiveness of Fermat test

- If the test outputs `composite` then n is definitely composite
- The test can can output `probable prime` if n is composite. In this case n is called a *pseudoprime*.
- There are some composite numbers for which the test will always output `probable prime` for every a with $\gcd(a, n) = 1$: these are called *Carmichael numbers*
- First few Carmichael numbers are: 561, 1105, 1729, 2465, ...

Carmichael Numbers

A Carmichael number *n* is a *composite* number that satisfies

$$b^{n-1} \equiv 1 \pmod{n},$$

for all integers *b*. Carmichael numbers constitute the (rare) instances where the converse of Fermat's theorem does not hold. There are infinitely many such numbers.

Outline

Chinese remainder theorem

Euler function ϕ

Testing for primality

- Fermat Test
- Miller–Rabin Test

Some Basic Complexity Theory

Factorisation problem

Discrete logarithm problem

Miller–Rabin test

- Same idea as Fermat test
- Can be guaranteed to detect composites if run sufficiently many times
- Most widely used test for generating large prime numbers

Square roots of 1

- A modular square root of 1 is a number x with $x^2 \bmod n = 1$
- When $n = pq$ there are 4 square roots of 1
- Two of these are 1 and -1 modulo n
- The other two are called *non-trivial* square roots of 1
- If x is a non-trivial square root of 1 then $\gcd(x-1, n)$ is a non-trival factor of n
- In other words, the existence of a non-trivial square root implies that n is composite

Miller–Rabin algorithm

Assume that n is odd and define u, v such that $n - 1 = 2^v u$, where u is odd

```
1. Pick $a$ randomly in the range $1 < a < n - 1$
2. Set $b = a^u \bmod n$
3. If $b == 1$ then return probable prime
4. For $j = 0$ to $v - 1$
    4.1 If $b == -1$ then return probable prime
    4.2 Else set $b = b^2 \bmod n$
5. Return composite
```

Note that when any output is returned the algorithm halts

Effectiveness of Miller–Rabin

- If the test returns `composite` then *n* is composite
- If the test returns `probable prime` then *n* *may* be composite
- If *n* is composite the test returns `probable prime` with probability at most $1/4$
- Therefore we repeat the algorithm *k* times while the output is `probable prime`
- The *k*-times algorithm will output `probable prime` when *n* is composite with probability no more than $(1/4)^k$
- In practice error probability is far smaller
- There are no composites less than 341,550,071,728,321 which pass the test for the seven bases $a = 2, 3, 5, 7, 11, 13, 17$

Why Miller–Rabin works

- Consider the sequence $a^u, a^{2u}, \ldots, a^{2^{v-1}u}, a^{2^v u} \bmod n$, where a is random with $0 < a < n-1$
- Each number in this sequence, after the first, is the square of the previous number
- If n is prime then Fermat's theorem tells us that the final value, $a^{2^v u} \bmod n = 1$
- Therefore if n is prime then either $a^u \bmod n = 1$ or there is a square root of 1 somewhere in this sequence and this value must be -1
- If a non-trivial square root of 1 is found then n is composite.

Example

Let $n = 1729$ which is a Carmichael number. Then $n - 1 = 1728 = 2 \times 864 = 4 \times 432 = 8 \times 216 = 16 \times 108 = 32 \times 54 = 64 \times 27$. So $v = 6$ and $u = 27$.

1. Choose $a = 2$.
2. $b = 2^{27} \bmod 1729 = 645$.
3. Since $b \neq 1$ continue.
4. - Next $b = 645^2 \bmod n = 1065$
   - Next $b = 1065^2 \bmod n = 1$
   - Thus $b = -1$ will never occur.
5. The algorithm returns `composite`.

Note that 1065 is a non-trivial square root of 1 modulo 1729. Indeed $\gcd(1729, 1064) = 133$ is a factor of 1729 (see slide 20).

Generating large primes

The Miller–Rabin test can be used to generate large primes:

1. Choose a random odd integer r of the same number of bits as the required prime
2. Test if r is divisible by any of a list of small primes
3. Apply Miller–Rabin test with 5 random bases
4. If r fails any test then set $r := r + 2$ and return to step 2

Note

This *incremental* method does not produce completely random primes. To do so, start from step 1 if r fails in step 4. Both options are seen in practice.

Complexity theory in cryptology

Computational complexity provides a foundation for

- analysing the computational requirements of cryptographic algorithms
- studying the difficulty of breaking ciphers

We can consider two aspects of computational complexity:

- algorithm complexity - how long it takes to run a particular algorithm
- problem complexity - what is the best (known) algorithm to solve a particular problem

Algorithm complexity

- The computational complexity of an algorithm is measured by its time and space requirements as functions of the size of the input m
- A positive function $f(m)$ is typically expressed as an "order of magnitude" of the form $\mathcal{O}(g(m))$ where $g(m)$ is another positive function. This is called "big O" notation.
- We say

  $$f(m) = \mathcal{O}(g(m))$$

  if there exist constants $c > 0$ and m_0 such that $f(m) \leq c \cdot g(m)$ for $m \geq m_0$
- This means that g is, at least in the long run, an upper bound for f
- Speak of *asymptotic* behaviour

Big O notation, illustrated

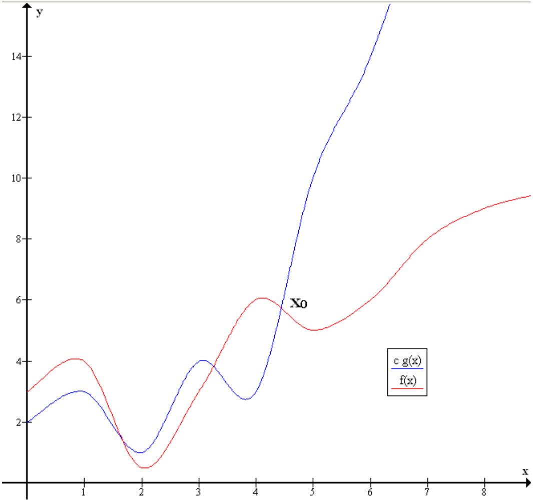


Image source: https://en.wikipedia.org/wiki/Big_O_notation

Polynomial and exponential functions

- A function $f(m)$ for which $f(m) = \mathcal{O}\left(m^t\right)$ for some positive integer t is said to be a *polynomial time function*.
- In cryptography we normally think of a polynomial time function as *efficient*.
- A function $f(m)$ for which $f(m) = \mathcal{O}\left(b^m\right)$ for some number $b > 1$ is said to be an *exponential time function*.
- In cryptography we normally think of a problem whose best solution is an exponential time function as *hard*
- Brute force key search is *exponential* as a function of the key length: an m-bit key length allows 2^m keys

Examples of algorithm complexity

1. If $f(m) = 17m + 10$ then

$$f(m) = \mathcal{O}\left(m\right)$$

since $17m + 10 \leq 18m$ for $m \geq 10$

2. If $f(m)$ is a polynomial:

$$f(m) = a_0 + a_1 \cdot m + \ldots + a_t m^t$$

then

$$f(m) = \mathcal{O}\left(m^t\right)$$

3. If $f(m) = \mathcal{O}\left(m^t\right)$ then it is also true that $f(m) = \mathcal{O}\left(m^{t+1}\right)$

Problem complexity

A problem is classified according to the minimum time and space needed to solve the hardest instances of the problem on a deterministic computer

1. Multiplication of two $m \times m$ matrices, with fixed size entries, using the obvious algorithm is $\mathcal{O}\left(m^3\right)$
2. Sorting a set of integers into ascending order is $\mathcal{O}\left(m \cdot \log_2 m\right)$ with algorithms such as Quicksort

Two important problems

1. **Integer factorisation**: given an integer, find its prime factors
2. **Discrete logarithm problem** (with base g): given a prime p and an integer y with $0 < y < p$, find x such that

$$y = g^x \bmod p$$

- Best known algorithms to solve these problems on conventional computers are *sub-exponential:* slower than polynomial but faster than any exponential
- Fast algorithms exist using *quantum computers*

Integer factorisation

- Factorisation by trial division is an exponential time algorithm and is hopeless for numbers of a few hundred bits
- A number of special purpose methods exist, which apply if the integer to be factorised has special properties
- The best current general method is known as the *number field sieve*
- The number field sieve is a *sub-exponential* time algorithm

Some factorisation records

Decimal digits	Bits	Date	CPU years
140	467	Feb 1999	?
155	512	Aug 1999	?
160	533	Mar 2003	2.7
174	576	Dec 2003	13.2
200	667	May 2005	121
232	768	Dec 2009	2000
240	795	Dec 2019	900
250	829	Feb 2020	2700

- All records used number field sieve
- The records are for numbers with only two large factors, so-called RSA numbers

Discrete logarithm problem (DLP)

Let $\mathbb{G}$ be a cyclic group with generator g. The *discrete log problem* (DLP) in $\mathbb{G}$ is:

$$\boxed{\text{given } y \text{ in } \mathbb{G}\text{, find } x \text{ with } y = g^x}$$

- The best known algorithm for solving DLP in $\mathbb{Z}_p^*$ is a variant of the *number field sieve* (also used for factorisation) — a *subexponential* algorithm in the length of p
- The DLP can also be defined on elliptic curve groups (see later lecture)
- Best known DLP algorithms on elliptic curves are *exponential*

Example in $\mathbb{Z}_{19}^*$

$g^x \bmod p$	x	$g^x \bmod p$	x
1	18	10	17
2	1	11	12
3	13	12	15
4	2	13	5
5	16	14	7
6	14	15	11
7	6	16	4
8	3	17	10
9	8	18	9

- Integers mod 19: $\mathbb{Z}_{19}^*$
- Generator $g = 2$
- When $y = g^x \bmod p$ then $\log_g y = x$
- For example, $\log_2 3 = 13$

Comparing brute-force key search, factorisation and discrete log in $\mathbb{Z}_p^*$

Symmetric key length	Length of $n = pq$	Length of prime p in $\mathbb{Z}_p^*$
80	1024	1024
112	2048	2048
128	3072	3072
192	7680	7680
256	15360	15360

- For example, brute force search of 128-bit keys for AES takes roughly same computational effort as factorisation of 3072-bit number with two factors of roughly equal size, or finding discrete logs in $\mathbb{Z}_p^*$ with a p of length 3072
- Source: NIST SP 800-57 Part 1 (2016)

Lecture 9: Public Key Cryptography and RSA

TTM4135

Relates to Stallings Chapter 9

Spring Semester, 2025

Motivation

- Public key cryptography (PKC) provides some features which cannot be achieved with symmetric key cryptography
- PKC is widely applied for key management in protocols such as TLS and IPSec
- RSA is probably the best known public key cryptosystem, widely deployed in many kinds of applications

Outline

Public Key Cryptography

RSA algorithms

Implementing RSA

Security of RSA
- Factorisation
- Side channel attacks

One-way functions

- A function f is said to be a *one-way function* if it is easy to compute $f(x)$ given x, but is computationally hard to compute $f^{-1}(y) = x$ given y
- It is an open problem in computer science whether any one-way functions formally exist
- Two examples of functions believed to be one-way are:
  1. Multiplication of large primes: the inverse function is integer factorisation
  2. Exponentiation: the inverse function is taking discrete logarithms

Trapdoor one-way functions

- A *trapdoor one-way function* f is a one-way function such that given additional information (the trapdoor) it is easy to compute f^{-1}
- An example of a trapdoor one-way function is *modular squaring*
- Let $n = pq$ be the product of two large prime numbers p and q and define $f(x) = x^2 \bmod n$
- If there is an algorithm to take square roots (compute f^{-1}) then this algorithm can be used to factorise n
- The trapdoor is the factorisation of n – knowledge of p and q gives an efficient algorithm to find square roots (exercise)

Ciphers based on computationally hard problems

- In 1976 Diffie and Hellman published their famous paper *New Directions in Cryptography*
- They suggested that computational complexity be applied in the design of encryption algorithms
- A public key cryptosystem can be designed by using a trapdoor one-way function
- The trapdoor will become the decryption key

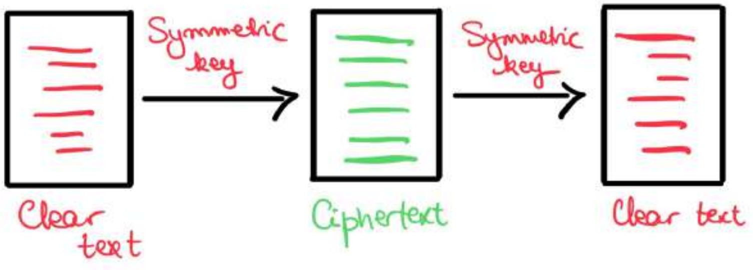
Prior claims

- In 1997 is was revealed that researchers at UK's intelligence agency (GCHQ) had previously invented public key cryptography in the early 1970s
- James H. Ellis, Clifford Cocks, and Malcolm Williamson invented what is now known as Diffie-Hellman key exchange and also a special case of RSA
- The GCHQ cryptographers used the name *non-secret encryption*

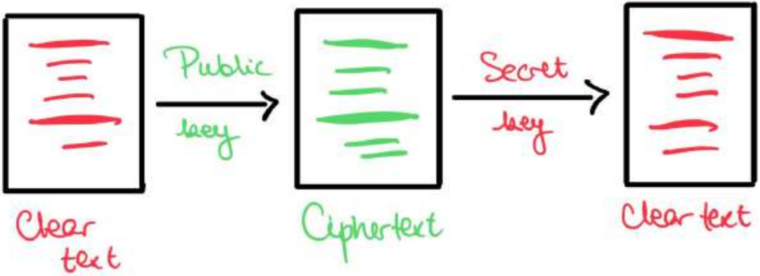
Public and private keys

- Public key cryptography is another name for *asymmetric cryptography*
- The encryption and decryption keys are different
- The encryption key is a *public key* which can be known to anybody
- The decryption key is a *private key* which should be known only to the owner of the key
- Finding the private key from knowledge of the public key must be a hard computational problem

Symmetric-key Cryptography – reminder



Public-key Cryptography – reminder



Why public key cryptography?

- Public key cryptography has two main advantages in comparison with shared key (symmetric key) cryptography

1. The key management is simplified: keys do not need to be transported confidentially
2. Digital signatures can be obtained. We look at digital signatures in a later lecture

Using public key encryption

- In a public key encryption scheme the receiver's key is made public
- Suppose that user A stores her public key, PK_A, in a public directory
- Anyone can obtain this public key and use it to encrypt a message M for A: $C = E(M, PK_A)$
- Since only A has the private key, SK_A, only A can decrypt and recover the message: $M = D(C, SK_A)$

Hybrid encryption

- Public key cryptography is usually computationally much more expensive than symmetric-key cryptography
- A typical usage of public key cryptography is to:
  - encrypt a random key for a symmetric-key encryption algorithm
  - encrypt the message M using the symmetric-key algorithm

  1. B chooses a random symmetric key k, finds A's public key PK_A and computes $C_1 = E(k, PK_A)$
  2. B computes $C_2 = E_s(M, k)$ where E_s is encryption with a symmetric-key algorithm, such as AES in CTR mode
  3. B sends (C_1, C_2) to A
- On receipt of (C_1, C_2), A recovers $k = D(C_1, SK_A)$ and then $M = D_s(C_2, k)$

Introduction to RSA

- Rivest-Shamir-Adleman, MIT, 1977
- Public-key cryptosystem and digital signature scheme
- Based on integer factorisation problem
- RSA patent expired in 2000

RSA Key Generation

1. Let p, q be distinct prime numbers, randomly chosen from the set of all prime numbers of a certain size
2. Compute $n = pq$
3. Select e randomly with $\gcd(e, \phi(n)) = 1$
4. Compute $d = e^{-1} \bmod \phi(n)$
5. The public key is the pair n and e
6. The private key consists of the values p, q and d

RSA operations

Encryption The public key for encryption is $K_E = (n, e)$

1. Input is any value M where $0 < M < n$
2. Compute $C = E(M, K_E) = M^e \bmod n$

Decryption The private key for decryption is $K_D = d$ (we will see later how to use values p and q)

1. Compute $D(C, K_D) = C^d \bmod n = M$

Note that any message needs to be pre-processed to become the input M: this includes coding as a number and adding randomness (details later)

Numerical example

- ***Key Generation***:
  - Suppose $p = 43, q = 59$ then $n = pq = 2537$ and $\phi(n) = (p-1)(q-1) = 2436$
  - Choose $e = 5$ then

$$d = e^{-1} \bmod \phi(n) = 5^{-1} \bmod 2436 = 1949$$

- ***Encryption***:
  Let $M = 50 \implies C = M^5 \bmod 2537 = 2488$
- ***Decryption***:
  $M = C^{1949} \bmod 2537 = 50$

Correctness of RSA Encryption (blackbaord)

We need to know that encryption followed by decryption gets back where we started from:

$$(M^e)^d \bmod n = M$$

Since $d = e^{-1} \bmod \phi(n)$ we know that $ed \bmod \phi(n) = 1$ and so $ed = 1 + k\phi(n)$ for some integer k. Therefore:

$$\begin{aligned} (M^e)^d \bmod n &= M^{ed} \bmod n \\ &= M^{1+k\phi(n)} \bmod n \end{aligned}$$

To complete the proof we need to show

$$M^{1+k\phi(n)} \bmod n = M \tag{1}$$

Proving equation 1: Case 1

There are two cases. We first assume $\gcd(M, n) = 1$
We can apply Euler's theorem directly to get

$$M^{\phi(n)} \bmod n = 1$$

Therefore

$$\begin{aligned} M^{1+k\phi(n)} \bmod n &= M \times (M^{\phi(n)})^k \bmod n \\ &= M \times (1)^k \bmod n \\ &= M \end{aligned}$$

Proving equation 1: Case 2

- If $\gcd(M, n) \neq 1$ then it must be the case that either $\gcd(M, p) = 1$ or $\gcd(M, q) = 1$
- Suppose that $\gcd(M, p) = 1$ (the other case is similar) Then $\gcd(M, q) = q$ so $M = lq$ for some integer l
- Applying Fermat's theorem we obtain $(M^{\phi(n)})^k \bmod p = (M^{p-1})^{(q-1)k} \bmod p = 1$ Therefore

$$M^{1+k\phi(n)} \bmod p = M \bmod p \qquad (2)$$

- Since $M = lq$ it follows that

$$M^{1+k\phi(n)} \bmod q = 0 \qquad (3)$$

Case 2 continued

- Finally the Chinese Remainder Theorem tells us that there is a unique solution $x \bmod n$ to the two equations (2) and (3) where $x = M^{1+k\phi(n)}$
- The solution $x = M$ satisfies both equations (2) and (3) and therefore this is the unique solution for $M^{1+k\phi(n)} \bmod n$
- Thus equation (1) is satisfied in this case too

RSA applications

The RSA operations can be used in a variety of applications.

- In this lecture we consider only message encryption.
- We look at RSA digital signatures in a later lecture.
- RSA is often used to distribute a key for symmetric-key encryption (often known as *hybrid encryption*).
- RSA can be used for user authentication by proving knowledge of the private key corresponding to an authenticated public key.

Implementation issues

Optimisations in the implementation of RSA have been widely studied. We examine some of the most important issues:

- key generation
  - choice of e
  - generating large primes
- encryption and decryption algorithms
  - fast exponentiation
  - using CRT for decryption
- formatting data (padding)

Generating p and q

- The primes p and q should be random of a chosen length. Today this length is usually recommended to be at least 1024 bits.
- A simple method of selecting a random prime is given by the following algorithm:
  1. Select a random odd number r of the required length.
  2. Check whether r is prime
  3. - If so, output r and halt
     - If not, increment r by 2 and go to the previous step.
  4. We require a fast way to check for primality such as the Miller–Rabin test.

Are there enough prime numbers?

- The *prime number theorem* tells us that the primes thin out as the numbers get larger.
- Let $\pi(x)$ denote the number of prime numbers less than x. The prime number theorem say that the ratio of $\pi(x)$ and $\frac{x}{\ln(x)}$ tends to 1 as x gets large.
- We can use the prime number theorem to give a rule of thumb that the proportion of prime numbers up to size x is $\ln(x)$.
- Since $\ln(2^{1024}) = 710$ we can estimate that one in every 710 numbers of size 1024 bits is a prime number. Therefore there are well over 2^{1000} 1024-bit primes.
- Thus brute-force searching for randomly chosen primes is completely infeasible.

Selecting e

- The public exponent e should be chosen at random for best security
- A small value of e is often used in practice since it can have a large effect on efficiency.
  - $e = 3$ is the smallest possible value and is sometimes used. However, there are possibly security problems when encrypting small messages.
  - $e = 2^{16} + 1$ is a popular choice. More exponentiations, but reduces the constraints on p and q, and avoids aforementioned attacks.
- A smaller than average d value is also possible. However, to avoid known attacks d should be at least $\sqrt{n}$
  - A low value of d implies a total break, since one can just brute-force all possible values.

Fast exponentiation

- To compute the RSA encryption and decryption functions we use the *square-and-multiply* modular exponentiation algorithm.
- We write e in binary representation.

$$e = e_0 2^0 + e_1 2^1 + \cdots + e_k 2^k$$

  where e_i are bits.
- The basic idea behind fast exponentiation is the *square and multiply* algorithm.
- There are many variants and optimisations of the basic idea.

Square and multiply algorithm

$$m^e = m^{e_0}(m^2)^{e_1}(m^4)^{e_2}\dots(m^{2^k})^{e_k}$$

```
Data: $m, n, e = e_k e_{k-1} \dots e_1 e_0$
Result: $m^e \bmod n$
$z \leftarrow 1$;
for $i = 0$ to $k$ do
    if $e_i = 1$ then
        $z \leftarrow z * m \bmod n$;
    end
    if $i < k$ then
        $m \leftarrow m^2 \bmod n$;
    end
end
return $z$
```

Algorithm 1: Square and multiply algorithm

Cost of square and multiply

- If $2^k \leq e < 2^{k+1}$ then the algorithm uses k squarings. If b of the e_i bits are 1 then the algorithm uses $b - 1$ multiplications. Note that the first computation $z \to z * m$ is not counted because then $z = 1$.
- Suppose that n is a 2048-bit RSA modulus. The public exponent e is length at most 2048 bits. To compute $M^e \mod n$ requires at most:
  - 2048 modular squarings; and
  - 2048 modular multiplications.
- On average only half of the bits of e are '1' bits and so only 1024 multiplications are needed.
- Remember that we can reduce modulo n after every operation.

Faster decryption with the CRT

- We can use the Chinese Remainder Theorem to decrypt ciphertext C faster with regard to p and q separately.
- First compute:

$$\begin{aligned} M_p &= C^{d \bmod p-1} \bmod p \\ M_q &= C^{d \bmod q-1} \bmod q \end{aligned}$$

- Solve for $M \bmod n$ using the Chinese remainder theorem.

$$\begin{aligned} M &\equiv M_p \pmod p \\ M &\equiv M_q \pmod q \end{aligned}$$

$$M = q \times (q^{-1} \bmod p) \times M_p + p \times (p^{-1} \bmod q) \times M_q \bmod n$$

Why it works (blackbaord)

Note that $d = d \bmod (p-1) + k(p-1)$ for some k.

$$\begin{aligned} M \bmod p &= (C^d \bmod n) \bmod p \\ &= C^d \bmod p \\ &= C^{d \bmod p-1} C^{k(p-1)} \bmod p \\ &= C^{d \bmod p-1} \\ &= M_p \end{aligned}$$

- Similarly $M \bmod q = M_q$
- Therefore $M \bmod n$ is the unique solution to the above two equations.

Example

- Same example as before: $n = 43 \times 59$. Ciphertext is $C = 2488$. Decryption exponent is $d = 1949$.
- $d \bmod p - 1 = 1949 \bmod 42 = 17$
  $d \bmod q - 1 = 1949 \bmod 58 = 35$
- $$\begin{aligned} M_p &\equiv 2488^{17} \pmod{43} = 37^{17} \pmod{43} = 7 \\ M_q &\equiv 2488^{35} \pmod{59} = 16^{35} \pmod{59} = 50 \end{aligned}$$
- Using CRT solution is $M = 50$.

How much faster is decryption with the CRT?

- Note that the exponents ($d \bmod p-1$) and ($d \bmod q-1$) are about half the length of d.
- Since the complexity of exponentiation (square and multiply) increases with the cube of the input length, computing M_p and M_q each use 1/8 the computation of computing $M = C^d \bmod n$.
- Overall there is about 4 times less computation. If M_p and M_q can be computed in parallel the time can be up to 8 times faster.
- This is a good reason to store p and q with the private exponent d.

RSA Padding

- Using the RSA encryption function directly on messages encoded as numbers is a weak cryptosystem. It is vulnerable to attacks such as:
  - building up a dictionary of known plaintexts
  - guessing the plaintext and checking to see if it encrypts to the ciphertext
  - Håstad's attack (next slide)
- Therefore padding mechanisms must be used to prepare messages for encryption. These mechanisms must include redundancy and randomness.

Håstad's Attack

- Suppose that the *same message* is encrypted without padding to three different recipients.
- Suppose that public exponent $e = 3$ is used by all recipients
- Then the cryptanalyst has three ciphertexts:

$$\begin{aligned} c_1 &= m^3 \bmod n_1 \\ c_2 &= m^3 \bmod n_2 \\ c_3 &= m^3 \bmod n_3 \end{aligned}$$

- These equations can be solved by the Chinese Remainder Theorem to obtain m^3 in the ordinary (non-modular) integers. Then m can be found by taking a cube root.

Types of padding

- PKCS #1: simple, ad-hoc design for encryption and signatures
- Optimal Asymmetric Encryption Padding (OAEP) designed by Bellare and Rogaway in 1994.
  - Has a security proof in a suitable model
  - Standardised in IEEE P1363: Standard Specifications for Public Key Cryptography

Example RSA block format: PKCS Number 1

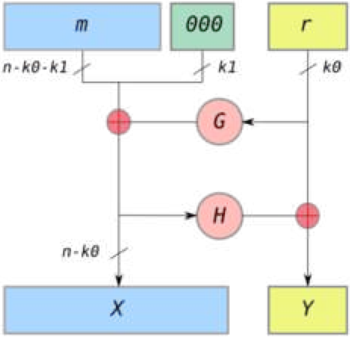
Encryption block format is:

00	02	*PS*	00	*D*

where 00 and 02 are bytes, *PS* is a pseudo-random string of nonzero bytes, and *D* is the data to be encrypted.

- The length of the block is the same as the length of the modulus.
- PS is a minimum of 8 bytes,
- The byte 02 and padding ensure that even short messages result in a large integer value for encryption.

Optimal Asymmetric Encryption Padding (OAEP)



Picture from Wikipedia

- The OAEP scheme includes k_0 bits of randomness and k_1 bits of redundancy into the message before encryption.
- Reasonable values of k_0 and k_1 are 128.
- Two random hash functions G and H are used
- Note that OAEP is an encoding algorithm - it can be easily inverted without any secret.

Outline

Public Key Cryptography

RSA algorithms

Implementing RSA

Security of RSA
- Factorisation
- Side channel attacks

Factorising the RSA modulus

- If an adversary can factorise the modulus n into its prime factors p and q then the adversary can easily recover the private key d and reveal all messages. Thus breaking RSA is not harder than the factorisation problem.
- Using a formal definition of encryption security it can be shown that breaking RSA is as hard as the so-called *RSA problem*.
- It is unknown whether the RSA problem is as hard as the factorisation problem. Remember that it is also unknown whether factorisation is really computationally hard.
- One positive security result is that finding the private key from the public key is as hard as factorising the modulus.

Equivalence with factorisation problem

- Is it possible to find the private key without factorising the modulus? No!

Theorem (Miller)

Determining d from e and n is as hard as factorising n.

- To show this, suppose that a cryptanalyst can find d from e and n.
- Then cryptanalyst could factorise n using Miller's algorithm (next slide).
- Algorithm uses same ideas as Miller–Rabin test for primality.

Miller's Algorithm

- Define u, v such that $ed - 1 = 2^v u$, where u is odd
- Consider the sequence $a^u, a^{2u}, \ldots, a^{2^{v-1}u}, a^{2^v u} \pmod{n}$, where a is random with $0 < a < n$.
- Notice that $a^{2^v u} \equiv a^{ed-1} \equiv a^{ed} a^{-1} \equiv aa^{-1} \equiv 1 \pmod{n}$. Therefore there is a square root of 1 somewhere in this sequence.
- With probability at least $\frac{1}{2}$ the sequence contains a non-trivial square root of 1 modulo n, thereby revealing the factors of n.
- If not, choose a new a and repeat.

Quantum computers

- Quantum computers do not exist yet (commercially at least).
- Shor's algorithm can factorise in polynomial time on a quantum computer.
- NIST is currently running an open competition to standardise signature schemes against quantum computers, and the standardisation process for signature schemes was (partly) finalised in 2023.

https://csrc.nist.gov/projects/pqc-dig-sig/round-1-additional-signatures
https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization

Outline

Public Key Cryptography

RSA algorithms

Implementing RSA

Security of RSA

- Factorisation
- Side channel attacks

Side Channel Attacks

- First made public in 1996 by Paul Kocher
- Many different kinds of side-channels are now known including:

Timing attacks Uses timing of the private key operations to obtain information about the private key.

Power analysis Uses power usage profile of the private key operations to obtain information about the private key.

Fault analysis Measures the effect of interfering with the private key operations to obtain information about the private key.

Timing attacks

- Recall that the square-and-multiply algorithm performs either a squaring or a squaring and a multiplication in each step
- The multiplication step is included exactly when each exponent bit $e_i = 1$
- Thus step i takes around twice as long when $e_i = 1$ as when $e_i = 0$

Demonstrated practically.

Some side channel countermeasures

- computing in constant time - run a “dummy” multiplication when $e_i = 0$
- Montgomery ladder - makes every operation depend on the key to avoid some fault attacks
- randomising the RSA message - mitigates “differential” attacks by preventing multiple timings on the same operation

Practical problems with RSA key generation

- In 2008 it was discovered that the implementation of OpenSSL used in Debian-based linux system used massively reduced randomness for RSA key generation.
- In 2012 a group of researchers led by Arjen Lenstra published a study of over 6 million RSA keys deployed on the Internet (many have expired).
  - 270 000 keys (about 4%) were identical, causing potential problems for those that share keys.
  - 12934 (about 0.2% of keys examined) provide no security because they share one prime factor with each other.
  - These problems are almost certainly due to poor random number generation.

Summary of RSA encryption security

- Standardised padding should always be used before encryption.
- Factorisation of the modulus is the best known attack against RSA in the case that standardised padding is used.
- Finding the private key from the public key is as hard as factorising the modulus.
- It is an open problem whether there is any way of breaking RSA encryption without factorising the modulus.
- Side channels

Lecture 10: Public Key Cryptosystems based on Discrete Logarithms

TTM4135

Relates to Stallings Chapter 10

Spring Semester, 2025

Motivation

- Discrete log ciphers are currently the main alternative public key systems to RSA
- Discrete log ciphers are widely deployed and standardised
- When implemented on elliptic curves, discrete log ciphers are often more efficient than RSA

Outline

Diffie–Hellman Key Exchange
- Protocol
- Properties

Elgamal Cryptosystem
- Algorithm
- Security

Elliptic Curves

Elliptic Curve Cryptography

Post-Quantum Cryptography

Diffie–Hellman key exchange

- Two users, Alice and Bob, want to share a secret using only public communications
- Public knowledge: generator g of a multiplicative group G of order t
- Alice and Bob each select random values a and b respectively where $0 < a, b < t$
- Alice sends g^a to Bob (*over an insecure channel*)
- Bob sends g^b to Alice (*over an insecure channel*)
- Alice and Bob both compute secret key $Z = g^{ab}$
- Originally group G used was $\mathbb{Z}_p^*$ for large p
- Today common to use elliptic curve group for G (see later in this lecture)

Diffie-Hellman Protocol

$$\begin{array}{ccc} \textbf{Alice} & & \textbf{Bob} \\ \text{Choose } a & & \\ & \xrightarrow{\quad g^a \quad} & \\ & & \text{Choose } b \\ & \xleftarrow{\quad g^b \quad} & \\ Z = (g^b)^a & & Z = (g^a)^b \end{array}$$

The shared secret value Z can be used to compute a key for, say, AES. This is done using a *key derivation function* based on a public hash function.

Example in $\mathbb{Z}_p^*$

Public knowledge is $p = 181, g = 2$

- Alice selects $a = 50$
- Bob selects $b = 33$
- Alice sends $g^{50} \bmod 181 = 116$ to Bob
- Bob sends $g^{33} \bmod 181 = 30$ to Alice
- Alice computes $Z = 30^{50} \bmod 181$
- Bob computes $Z = 116^{33} \bmod 181$
- Common secret is $Z = 49$

Security of Diffie–Hellman

- An attacker who can find discrete logarithms in G can break the protocol:
  1. intercept the value g^a and take the discrete log to obtain a
  2. compute g^{ab} in the same way as B
- It is unknown whether a better way exists to break the protocol than by taking discrete logs
- The problem of finding $Z = g^{ab}$ from knowledge of g^a and g^b is known as the *Diffie–Hellman problem*

Authenticated Diffie–Hellman

- In the basic Diffie–Hellman protocol the messages between Alice and Bob are not authenticated
- Neither Alice nor Bob knows who the secret Z is shared with unless the messages are authenticated
- This allows a *man-in-the-middle* attack, where the adversary sets up two keys, one with Alice and one with Bob, and relays messages between the two
- Authentication can be added in different ways, for example by adding *digital signatures* (see next lecture for how to construct digital signatures)

Static and ephemeral Diffie–Hellman

- The Diffie–Hellman protocol described above uses *ephemeral keys*: keys which are used once and then discarded
- In *static* Diffie–Hellman, Alice chooses a long-term private key x_A with corresponding public key $y_A = g^{x_A}$
- Similarly, Bob chooses a long-term private key x_B with corresponding public key $y_B = g^{x_B}$
- Now Alice and Bob can find a shared secret $S = g^{x_A x_B}$ just by looking up (or knowing beforehand) each others' public keys
- The secret S is static: it stays the same long-term, until Alice and Bob change their public keys

Elgamal cryptosystem

- Proposed by Taher Elgamal in 1985
- Turns the Diffie–Hellman protocol into a cryptosystem
- Here we look at original version where group G is $\mathbb{Z}_p^*$
- Alice combines her ephemeral private key with Bob's long-term public key

Elgamal key generation

- Select a prime p and a generator g of $\mathbb{Z}_p^*$
- Select a long term private key x where $1 < x < p - 1$
- Compute $y = g^x \bmod p$
- The public key is (p, g, y)
- Often p and g are shared between all users in some system

Encryption and decryption

Encryption The public key is $K_E = (p, g, y)$

1. For any value M where $0 < M < p$
2. Choose k at random and compute $g^k \bmod p$
3. $C = E(M, K_E) = (g^k \bmod p, My^k \bmod p)$

Decryption The private key is $K_D = x$ with $y = g^x \bmod p$

1. Let $C = (C_1, C_2)$
2. Compute $C_1^x \bmod p$
3. $D(C, K_D) = C_2 \cdot (C_1^x)^{-1} \bmod p = M$

Why it works

- The sender knows the ephemeral private key k and the long-term public key y
- The receiver knows the static private key x and the ephemeral public key $g^k \bmod p$
- Both sender and recipient can compute the Diffie–Hellman value for the two public keys: $C_1 = g^k \bmod p$ and $y = g^x \bmod p$
- This Diffie–Hellman value, $y^k \bmod p = C_1^x \bmod p$, is used as a mask for the message M

Powers of integers modulo 19

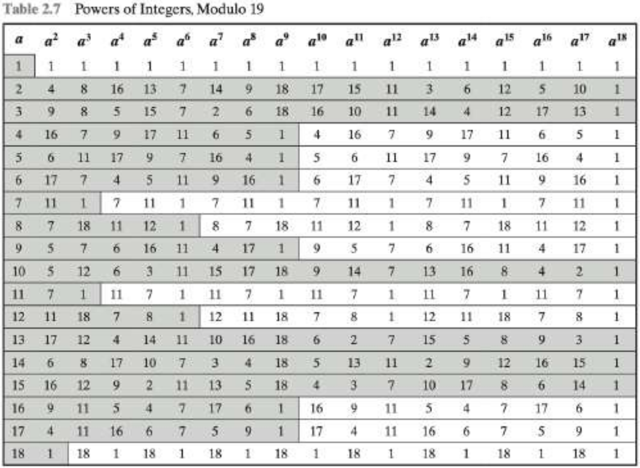


Table 2.7 Powers of Integers, Modulo 19

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1
3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	1
4	16	7	9	17	11	6	5	1	4	16	7	9	17	11	6	5	1
5	6	11	17	9	7	16	4	1	5	6	11	17	9	7	16	4	1
6	17	7	4	5	11	9	16	1	6	17	7	4	5	11	9	16	1
7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	7	18	11	12	1	8	7	18	11	12	1	8	7	18	11	12	1
9	5	7	6	16	11	4	17	1	9	5	7	6	16	11	4	17	1
10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	1
11	7	1	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	11	18	7	8	1	12	11	18	7	8	1	12	11	18	7	8	1
13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	1
14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	1
15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	1
16	9	11	5	4	7	17	6	1	16	9	11	5	4	7	17	6	1
17	4	11	16	6	7	5	9	1	17	4	11	16	6	7	5	9	1
18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1

Stallings book, Table 2.7

Example

Key generation Choose prime $p = 181$ and generator $g = 2$

- Private key of A is $x = 50$
- Public key is $p = 181, g = 2, y = 116$

Encryption Sender wants to send $M = 97$

- Sender chooses random $k = 31$
- Ciphertext is $(98, 173) = (C_1, C_2)$

Decryption A receives (C_1, C_2) and recovers M by:

- $C_1^x = 98^{50} \bmod p = 138$
- $M = C_2 \times (C_1^x)^{-1} \bmod p$
  $= 173 \times 138^{-1} \bmod 181$
  $= 97$

Security of Elgamal

- If an attacker can solve the discrete log problem, the system can be broken by determining the private key x from $g^x \bmod p$
- It is quite possible for many users to share the same p and g values.
- There is no need for any padding as in RSA - notice that each ciphertext is already randomised
  - For the encryption scheme itself – may be required for some other properties
- The Elgamal cryptosystem has a proof of security in a suitable model subject to the difficulty of the so-called *decision Diffie–Hellman problem*.

Discrete Log Problem over $\mathbb{Z}_p$

- Let p be a large prime.
- We will let g denote a generator of the multiplicative group of $\mathbb{Z}_p$
- i.e. for a any nonzero element in $\mathbb{Z}_p$ we can find a unique i between 1 and $p - 1$ such that $a = g^i \pmod{p}$.

We say that $\log_g(a) = i \pmod{p}$.

Discrete Log problem

The discrete log problem is:

$$\text{given } a\text{, find } \log_g(a) \pmod{p}.$$

For sufficiently large *p* this is an intractable problem.

Using Discrete Log

We shall show that several public key ciphers can be based on the discrete log problem.

It should be noted that the implementation of these ciphers depends on the property that $a^{p-1} \equiv 1$ for all nonzero elements in $\mathbb{Z}_p$.

- If it is possible to compute discrete logs in G, then Decision Diffie-Hellman does not hold.

Discrete Log Problem and Factoring

Solving the discrete log problem over $\mathbb{Z}_p$ is comparable to the difficulty of factoring n, where n is the product of two primes, i.e. if the number of bits in n is the same as the number of bits in p. Hence, the discrete log ciphers modulo p offer the same level of security as the RSA algorithm.

What are elliptic curves?

- Elliptic curves are algebraic structures formed from cubic equations
- An example is the set of all (x, y) pairs which satisfy the equation:

$$y^2 = x^3 + ax + b \bmod p$$

- This example is a curve over the field $\mathbb{Z}_p$ but elliptic curves can be defined over any field
- Once an identity element is added, a binary operation (like addition or multiplication) can be defined on these points
- With this operation the elliptic curve points form an *elliptic curve group*

An example

- Recall $y^2 = x^3 + ax + b \bmod p$. Call this curve $E_{23}(1, 1)$.
- Let $a = b = 1$, and $p = 23$, so we look at the curve $y^2 = x^3 + x + 1$.
- We can show that $x = 9$ and $y = 7$ satisfies the equation.
  - $7^2 = 49 \equiv 3 \bmod 23$.
  - $9^3 + 9 + 1 \equiv 3 \bmod 23$.
  - So then indeed, we have that $7^2 \equiv 9^3 + 9 + 1 \bmod 23$.
- Therefore, the point $P = (9, 7)$ is on the curve.

Other points on $E_{23}(1, 1)$

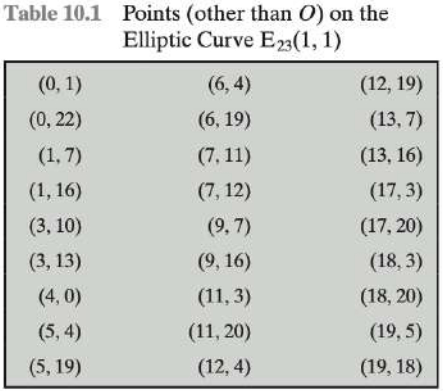


Table 10.1 Points (other than O) on the Elliptic Curve $E_{23}(1, 1)$

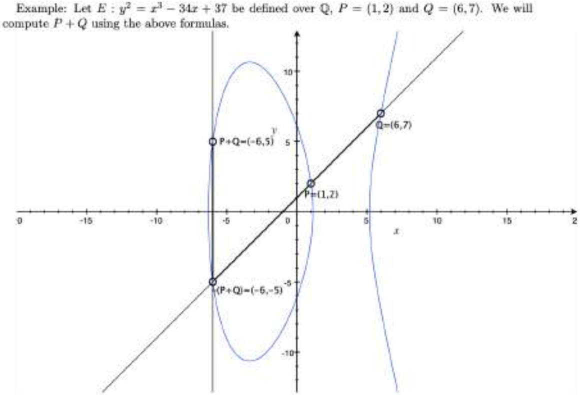
(0, 1)	(6, 4)	(12, 19)
(0, 22)	(6, 19)	(13, 7)
(1, 7)	(7, 11)	(13, 16)
(1, 16)	(7, 12)	(17, 3)
(3, 10)	(9, 7)	(17, 20)
(3, 13)	(9, 16)	(18, 3)
(4, 0)	(11, 3)	(18, 20)
(5, 4)	(11, 20)	(19, 5)
(5, 19)	(12, 4)	(19, 18)

Stallings book, Table 10.1

Elliptic curve computations

- The elliptic curve group operation could be denoted by any symbol, but by convention is it usually called elliptic curve *addition*
- We write $P + Q = R$ to show the group operation on curve points P and Q with result R
- The *elliptic curve discrete log problem* is to find the value of m, given a point P and a generator G so that $P = mG = G + G + \ldots + G$ (m times)
- Efficient computation of elliptic curve multiplication can use the *double-and-add algorithm*, by replacing multiplication in the square and multiply algorithm with addition

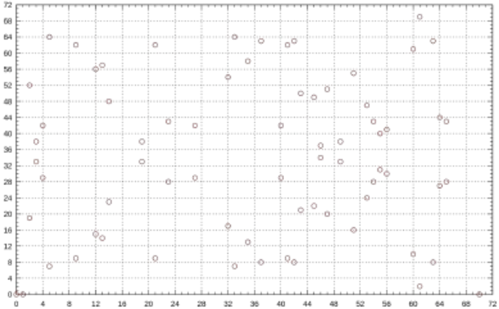
Elliptic Curve – visual representation



Example: Let $E : y^2 = x^3 - 34x + 37$ be defined over $\mathbb{Q}$, $P = (1,2)$ and $Q = (6,7)$. We will compute $P + Q$ using the above formulas.

https:
//www.umsl.edu/~siegelj/information_theory/projects/elliptic_curves_group_law.pdf

Elliptic Curve – visual representation



Representation of $y^2 = x^3 - x$ over the finite field $\mathbb{F}_{71}$
https://en.wikipedia.org/wiki/Elliptic_curve

Elliptic curve representations

There are several different ways of representing elliptic curves which use different point formats and different ways to computing the group operation

Short Weierstrass form is the most common format as shown on Slide 21

Montgomery form allows a fixed time elliptic curve multiplication which is useful to avoid timing side channels

Edwards form allows for faster group operations

It is common to shift between representations to optimise performance

Choosing elliptic curves

- A new elliptic curve can be generated at any time, but standard applications usually use standard curves
- Various predefined sets of curves exist such as the set of NIST curves contained in the standard SP 800-186 (2023)
- Desirable that standard curves are generated in a way to ensure there are no hidden special properties but researchers have disputed this for some standard curves

Example - NIST curve P-256

$$
\begin{aligned}
p &= \texttt{115792089210356248762697446949407573530086143415290314195533631308867097853951} \\
n &= \texttt{115792089210356248762697446949407573529996955224135760342422259061068512044369} \\
s &= \texttt{3045ae6fc8422f64ed579528d38120eae12196d5} \\
c &= \texttt{3099d2bbbfcb2538542dcd5fb078b6ef5f3d6fe2c745de65} \\
b &= \texttt{41058363725152142129326129780047268409114441015993725554835256314039467401291} \\
G_x &= \texttt{48439561293906451759052585252797914202 76294952604174799584408071708240463528 6} \\
G_y &= \texttt{36134250956749795798585127919587881956 61110667298501507187719825356841440510 9}
\end{aligned}
$$

- Curve of n points (256-bits) over $\mathbb{Z}_p$ with generator (G_x, G_y) and equation: $y^2 = x^3 - 3x + b \bmod p$
- Parameter s is the seed for the random generation and c is the output of a SHA-1 hash generated from s

Curve 25519

- A curve allowing very fast computations
- Proposed by Bernstein in 2005 and now a NIST recommended curve
- Equation of the curve is $y^2 = x^3 + 486662x^2 + x$ which is in Montgomery form
- The field for computations is the integers modulo p, where $p = 2^{255} - 19$ is prime

Discrete logarithms on elliptic curves

- The discrete logarithm on elliptic curve groups is to find the value of *m*, given a point *P* and a generator *G* so that $P = mG = G + G + \ldots + G$ (*m* times)
- This is the same as in $\mathbb{Z}_p^*$ but with addition as the group operation instead of multiplication
- The best known algorithms for solving the elliptic curve discrete log problem are *exponential* in the length of the parameters
  - Subexponential algorithms exist for factoring and finite field discrete log
- Consequently elliptic curve implementations use much smaller keys
- Compared with RSA the relative advantage of elliptic curve cryptography will *increase* at higher security levels

Comparing strength of elliptic curve cryptography

Symmetric key length	RSA modulus length or length of p in $\mathbb{Z}_p^*$	Elliptic curve group size
80	1024	160
128	3072	256
192	7680	384
256	15360	512

- For example, brute force search of 128-bit key for AES takes roughly same computational effort as for taking discrete logarithms in $\mathbb{Z}_p^*$ with p of 3072-bits or on an elliptic curve with elements of size 256 bits
- Source: adapted from NIST SP 800-57 Part 1, Recommendations for Key Management (revised 2020)

Elliptic curve cryptography

- Most cryptosystems based on discrete logarithms can be constructed with elliptic curves as well as in $\mathbb{Z}_p^*$
- In particular, Diffie–Hellman key exchange and Elgamal encryption can be run on elliptic curves
- Elliptic curve cryptography is today widely deployed
- The Canadian company Certicom, now part of Research in Motion, holds many patents on some of the mathematical ideas

Post-quantum cryptography

- Most public key cryptography is use today will be broken if quantum computers become available due to Shor's algorithm for factorisation, which can also be used to find discrete logarithms
- *Post-quantum cryptography* is concerned with building cryptographic primitives which will remain secure if this happens
- Symmetric key cryptography can still be used but with double length keys due to Grover's algorithm for searching
- NIST process to standardise post-quantum cryptography has recommended initial algorithms but is still continuing

Post-quantum Diffie–Hellman

- Post-quantum secure cryptosystems are based on different problems, particularly lattice problems, coding theory, and solving multi-variable polynomials
- Currently we do not have a post-quantum “drop-in” replacement for Diffie–Hellman
- A promising candidate was the use of isogenies on elliptic curves
    - But this was broken, see: `https://ellipticnews.wordpress.com/2022/08/12/attacks-on-sidh-sike/`

Lecture 11: Digital Signatures

TTM4135

Relates to Stallings Chapters 13

Spring Semester, 2025

Motivation

- Obtaining digital signatures is one of the major benefits of public key cryptosystems
- In some countries digital signatures are legally binding in the same way as handwritten signatures
- Digital signature are deployed widely in authentication protocols and management of public keys

Outline

Digital Signature Properties

Reminder

RSA Signatures

Discrete Logarithm Signatures

- Elgamal Signatures
- Schnorr Signatures
- DSA
- ECDSA

Confidentiality and authentication

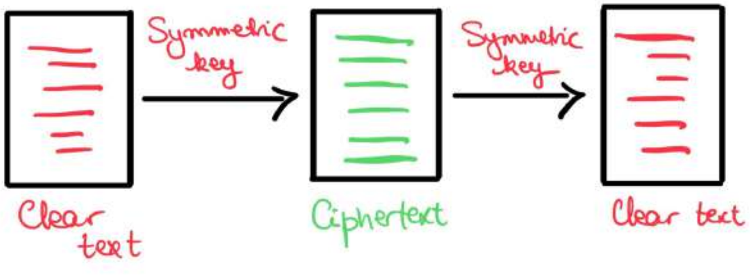
- Message authentication codes (MACs) allow only an entity with the shared secret to generate a valid MAC tag, providing data integrity and data authentication
- Digital signatures use public key cryptography to provide the properties of a MAC and more: only the owner of the private key can generate a correct digital signature
- Digital signatures provide the *non-repudiation* security service because a judge can decide which party formed the signature

Comparing physical and digital signatures

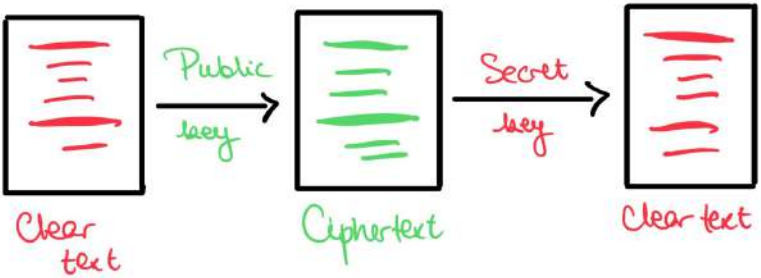
Physical signatures	Digital signatures
Produced by human	Produced by machine
Same on all documents	Function of message
Easy to recognise	Requires computer to check

Both types of signature need to be difficult to forge

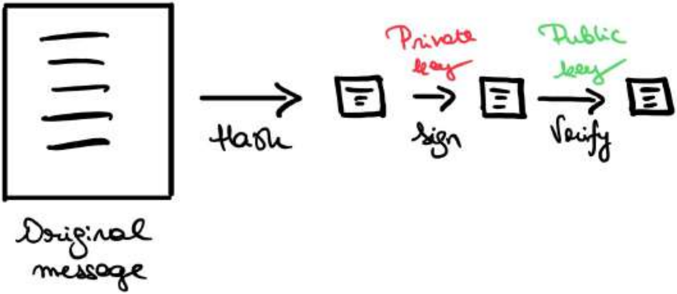
Symmetric Key Encryption



Public Key Encryption



Signatures



Digital signatures overview

- Digital signatures allow a *signer* Alice with a *public key K_V* to *sign* a message, using her *associated secret key K_S*, in such a way that *anyone who knows K_V can verify* the signature
- More precisely, anyone who knows K_V can verify:
  - The message originated from Alice
  - The message was not modified in transit

Digital signatures: an example

Ex: software company,disseminate a software update to clients.

- Should be possible for any of its clients to verify that the update is authentic.
- Shouldn't be possible for a malicious third party to convince the clients to accept an update that was not released by the company
  - Company can generate a private signing key K_S, along with a public verification key K_V.
  - Distribute K_V to its clients (e.g. could be a part of the original purchase bundle), keep K_S private.
  - When realeasing an update m, company computes a signature σ; sends (m, σ).
  - All clients who have K_V can verify the software update.
  - No malicious adversary should be able to make a client accept a fraudulent update (m', σ') under the key K_V (even if m' is close to m).

Comparison with MACs

- Public-key analogue of MACs
- Using digital signatures rather than MACs simplifies key distribution and management
- Especially when one sender needs to communicate with multiple receivers (as in the example in the previous slide)
- This way, the sender avoids having to set up one key per recipient
- Digital signatures are *publicly verifiable* and *transferable* (Bob can check Alice's signature, and pass it on to Eve who can also check)
- Digital signatures also provide *non-repudiation* – important eg in legal applications. This cannot be done with a symmetric key object
- MACs are shorter and 2-3x more efficient to generate/ verify

Elements of digital signatures

- A digital signature scheme has three algorithms:
  - key generation
  - signature generation
  - signature verification
- Key generation outputs two keys:
  - a private *signature generation key* or simply *signing key*, K_S
  - a public *signature verification key*, K_V

Signature generation algorithm

Suppose that Alice wishes to generate a signature on a message m

Inputs
- Alice's private *signing key*, K_S
- The message m

Output
- Signature $\sigma = \mathsf{Sig}(m, K_S)$

- Only the owner of the signing key should be able to generate a valid signature
- It should be possible to choose the message as any bit string

Signature verification algorithm

Suppose that Bob wishes to verify a claimed signature σ on the message *m*

Inputs
- Alice's public *verification key*, K_V
- The message *m*
- The claimed signature σ

Output
- A boolean value $\text{Ver}(m, \sigma, K_V) =$ true or false

- Anyone should be able to use the signature verification key to verify a valid signature

Properties required of verifying function

Correctness If $\sigma = \mathrm{Sig}(m, K_S)$ then $\mathrm{Ver}(m, \sigma, K_V) = \mathrm{true}$, for any matching signing/verification keys

Unforgeability It is computationally infeasible for anyone without K_S to construct m and σ such that $\mathrm{Ver}(m, \sigma, K_V) = \mathrm{true}$

- The signing algorithm Sig *may* be randomised so that there are many possible signatures for a single message
- For a stronger security definition we often assume that an attacker has access to a *chosen message* oracle: forging a (new) signature should be difficult even if the attacker can obtain signatures on messages of his choice

Security goals

An attacker may try to break a digital signature scheme in several ways

Key recovery The attacker attempts to recover the private signing key from the public verification key and some known signatures

Selective forgery The attacker chooses a message and attempts to obtain a signature on that message

Existential forgery The attacker attempts to forge a signature on any message not previously signed, even if it is a meaningless message

Modern digital signatures are considered secure only if they can resist *existential forgery under a chosen message attack*.

RSA signatures

- Proposed back in 1978 along with the RSA encryption scheme
- Still widely deployed, particularly where signatures are verified many times with a small public exponent
- Standardised in various places including the latest NIST FIPS186 standard (2023)
- Can be broken by an attacker who can factorise the modulus

RSA signatures - key generation

RSA signature keys are generated in the same way as RSA encryption keys

- A modulus $n = pq$ is computed from random large primes p and q
- Two exponents e and d are generated with

$$ed \bmod \phi(n) = 1$$

- Private signing key is $\mathtt{sk} = (d, p, q)$
- Public verification key is $\mathtt{pk} = (e, n)$
- A hash function h is also required and should be a fixed public parameter of the signature scheme

RSA signature operations

Signature generation: Inputs are the message m, the modulus n and the private exponent d

1. Compute signature $\sigma = h(m)^d \bmod n$

Signature verification: Inputs are the message m, the claimed signature σ and the public key (e, n)

1. Compute $h' = h(m)$
2. Check whether $\sigma^e \bmod n = h'$ and if so output true, otherwise output false

Hash functions for RSA signatures

- In the above signature definition, we could let *h* be a standard hash function, such as SHA-256
- The following two choices both can be proven secure with suitable assumptions
  1. A *full domain hash* is an implementation of *h* which can take values randomly in the range 1 to *n*
  2. PSS is a probabilistic hashing function similar to OAEP used for RSA encryption and is standardised in the PKCS #1 standard, available as RFC 8017

Discrete logarithm signatures

- These are digital signatures whose security relies on the difficulty of the discrete log problem
- We look at four versions
    1. Original Elgamal signatures from 1985 set in $\mathbb{Z}_p^*$
    2. Schnorr signatures
    3. The digital signature algorithm (DSA) standardised by NIST in the Digital Signature Standard. A highly optimized version of ElGamal signatures
    4. The version of DSA based on elliptic curve groups, known as ECDSA

Elgamal signature scheme in $\mathbb{Z}_p^*$

- Let p be a large prime, g a generator for $\mathbb{Z}_p^*$. Private signing key is x with $0 < x < p - 1$
- Public verification key is $y = g^x \bmod p$. Values p, g and y are public knowledge
- Suppose Alice wants to sign a value m which is an integer between 0 and $p - 1$

Signature operations

Signature generation To sign message m with signing key x:

1. Select random k, $0 < k < p - 1$ and compute $r = g^k \bmod p$
2. Compute $s = k^{-1}(m - xr) \bmod (p - 1)$
3. Signature is the pair $\sigma = (r, s)$

Signature verification Given message m and claimed signature $\sigma = (r, s)$ and verification key y:

1. Verify that $g^m \equiv y^r r^s \bmod p$

Schnorr signature scheme in $\mathbb{Z}_p^*$

- Let p be a large prime, g a generator for $\mathbb{Z}_p^*$. Private signing key is x with $0 < x < p - 1$
- Public verification key is $y = g^x \bmod p$. Values p, g and y are public knowledge
- Suppose Alice wants to sign a value m which is an integer between 0 and $p - 1$

Signature operations

Signature generation To sign message m with signing key x:

1. Select random k, $0 < k < p - 1$ and compute $r = g^k \bmod p$
2. Let $e = H(r||m)$
3. Compute $s = k - xe \bmod (p - 1)$
4. Signature is the pair $\sigma = (s, e)$

Signature verification Given message m and claimed signature $\sigma = (s, e)$ and verification key y:

1. Let $r_v = g^s y^e$
2. Let $e_v = H(r_v||m)$
3. Accept if $e_v = e$

Digital signature algorithm (DSA)

- First published by NIST in 1994 as FIPS PUB 186
- In the latest version of the NIST standard FIPS PUB 186-5 (February 2023) DSA is no longer recommended (can be used for verification in legacy applications)
- Based on Elgamal signatures
- Simpler calculations and shorter signatures due to restricting the calculations to a *subgroup* of $\mathbb{Z}_p^*$ or to an elliptic curve group
- Designed for use with the SHA family of hash functions
- Avoids some attacks that Elgamal signatures may be vulnerable to

DSA parameters

- p, a prime modulus of L bits
- q, a prime divisor of $p-1$ of N bits
- Valid combinations of L and N are: $(L = 1024, N = 160)$, $(L = 2048, N = 224)$, $(L = 2048, N = 256)$, $(L = 3072, N = 256)$
- $g = h^{\frac{p-1}{q}} \bmod p$, where h is any integer, $1 < h < p-1$
- H, the SHA hash family variant which outputs an N-bit digest
- NIST Special Publication SP 800-57 does *not* approve the first choice of parameters $(L = 1024, N = 160)$

Key and signature generation

Key generation Do this once at the start

1. Choose a random integer x with $0 < x < q$
2. x is the secret signing key
3. $y = g^x \bmod p$ is the public verification key

Signature generation For every message m to be signed

1. Choose k at random with $0 < k < q$ and set $r = (g^k \bmod p) \bmod q$
2. Set $s = k^{-1}(H(m) - xr) \bmod q$
3. The signature consists of the pair $\sigma = (r, s)$

Signature verification

A received signature of message m is a pair (r, s). To verify:

- Check that $0 < r < q$ and $0 < s < q$
- Compute $w = s^{-1} \bmod q$ and let:

$$\begin{aligned} u_1 &= H(m)w \bmod q \\ u_2 &= rw \bmod q \end{aligned}$$

- Check whether $(g^{u_1} y^{-u_2} \bmod p) \bmod q = r$

Comparison with Elgamal signatures

The verification equation is the same as for Elgamal signatures, except that all exponents are reduced modulo q and the final result is also reduced modulo q

- The complexity of signature generation is now mainly just one exponentiation with a short exponent (such as 224 or 256 bits)
- Signature verification requires two such short exponentiations
- The signature size is only $2N$ bits:
  - 448 bits when $N = 224$
  - 512 bits when $N = 256$

ECDSA

- Elliptic curve DSA (ECDSA) is defined in the standard FIPS 186-5
- Signature generation and verification is similar to DSA except that:
  - the parameter q becomes the order of the elliptic curve group
  - multiplication modulo p is replaced by the elliptic curve group operation
  - after the operation on the group elements only the x-coordinate (an element in the underlying field) is kept
- Description on following slides omits some of the detailed checks in FIPS186-5

ECDSA parameters

Parameters

- E an approved elliptic curve field and equation
- G the elliptic curve group generator, or base point
- n the order of the elliptic curve group and a prime number
- H the SHA-2 hash family variant which outputs an N-bit digest

Parameters are chosen from the NIST approved curves

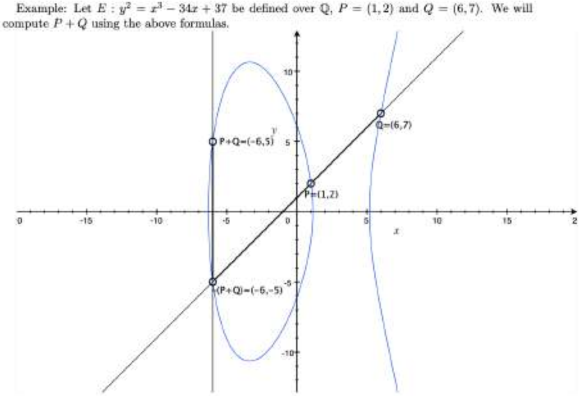
Interlude: reminder on elliptic curves, previous lecture

- Elliptic curves are algebraic structures formed from cubic equations
- An example is the set of all (x, y) pairs which satisfy the equation:
$$y^2 = x^3 + ax + b \bmod p$$
- This example is a curve over the field $\mathbb{Z}_p$ but elliptic curves can be defined over any field
- Once an identity element is added, a binary operation (like addition or multiplication) can be defined on these points
- With this operation the elliptic curve points form an *elliptic curve group*

Elliptic curve computations

- The elliptic curve group operation could be denoted by any symbol, but by convention is it usually called elliptic curve *addition*
- We write $P + Q = R$ to show the group operation on curve points P and Q with result R
- The *elliptic curve discrete log problem* is to find the value of m, given a point P and a generator G so that $P = mG = G + G + \ldots + G$ (m times)
- Efficient computation of elliptic curve multiplication can use the *double-and-add algorithm*, by replacing multiplication in the square and multiply algorithm with addition

Elliptic Curve – visual representation



Example: Let $E : y^2 = x^3 - 34x + 37$ be defined over $\mathbb{Q}$, $P = (1, 2)$ and $Q = (6, 7)$. We will compute $P + Q$ using the above formulas.

https://www.umsl.edu/~siegelj/information_theory/projects/elliptic_curves_group_law.pdf

ECDSA key generation

Key generation

1. Choose a random integer d with $0 < d < n$ as the secret signing key
2. Compute $Y = dG$ as the public verification key in the group G

The standard requires that before a public key is used it is checked to be a point on the curve G different from the identity

Signature generation

Signature generation For every message m to be signed

1. Let $e = H(m)$
2. Choose k at random with $0 < k < n - 1$ and compute $(x, y) = kG$
3. Set $r = x$, but return to step 2 if $r = 0$
4. Set $s = k^{-1}(e + rd) \bmod n$
5. The signature consists of the pair $\sigma = (r, s)$

Note that r is the x-coordinate of an elliptic curve point while s is an integer modulo n

Signature verification

A received signature of message m is a pair (r, s). To verify:

- Check that $0 < r < n$ and $0 < s < n$
- Compute $w = s^{-1} \bmod n$ and $e = H(m)$ and set:

$$\begin{aligned} u_1 &= ew \bmod n \\ u_2 &= rw \bmod n \end{aligned}$$

- Compute the point $(x, y) = u_1 G + u_2 Y$. The signature is valid if (x, y) is not the identity element in the curve E and

$$r \equiv x \pmod{n}$$

ECDSA variants

The new FIPS 186-5 standard includes other important signature schemes using elliptic curve groups

1. Deterministic ECDSA signatures
   - The per-message key *k* is deterministically computed as a function (based on HMAC) of the message to be signed and the private signing key *d*
   - Recommended when good random number generator is not available
2. EdDSA signatures
   - Uses the Edwards curve 25519
   - Deterministic version of Schnorr signatures

ECDSA vs. DSA

- Because of the clever design of DSA, signatures using ECDSA are generally no shorter than signatures using DSA for the same security level
- ECDSA signature size varies with the curve used. For approved curves this can vary from 448 bits to more than 1024 bits
- ECDSA public keys are shorter than DSA public keys

Lecture 12: Key Establishment and Certificates

TTM4135

Relates to Stallings Chapters 15 and 16

Spring Semester, 2025

Motivation

- So far, we have looked at cryptographic primitives (symmetric key encryption, public key encryption, signature schemes)
- Now, we are going to look at how to use these to build more advanced cryptographic protocols
- Key management is an essential part of setting up secure communications
- Key establishment is the process of setting up cryptographic keys to protect a subsequent communication session
- Key establishment in TLS uses public keys to allow clients and servers to share a new communication key
- Digital certificates are a vital tool for key establishment based on public key cryptography

Outline

Key establishment principles

Key establishment types
- Session Key Distribution using Symmetric Keys
- Session Key Distribution using Asymmetric Keys

Key establishment using public key encryption
- Certificates and PKI

Key establishment using symmetric key encryption
- Needham-Schroeder protocol
- Kerberos

Key management

Key management is concerned with generation, distribution, protection, destruction, of cryptographic keys. It is a critical aspect of any cryptographic system and includes several phases:

- Key generation: ideally keys are random
- Key distribution: keys must be distributed in a secure fashion
- Key storage: keys must be accessible for use but not to unauthorised users
- Key destruction: removing a key from memory is not always easy

Types of keys

Long-term keys: (or static keys) intended to be used for a long period – depending upon the application this may be a few hours or a few months or a few years

Ephemeral keys: generated for single use and then deleted, Diffie–Hellman being a typical example

Session keys: intended to be used for one communication session – depending upon the application this may be a few seconds, a few hours or a day

Types of keys and usage

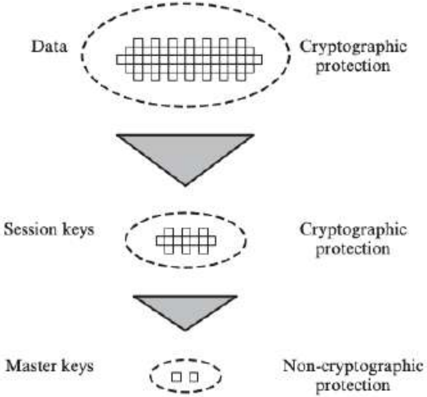


Figure 14.2 The Use of a Key Hierarchy

Stallings book, *seventh edition*. See Figure 15.2 in the *eight* edition.

Types of keys and usage

- Session keys are usually symmetric keys used to protect communications in a session with ciphers such as AES for authenticated encryption
- Session keys should make different sessions *independent* in the sense that compromise of one session key should not affect other sessions
- Long-term keys can be either symmetric or asymmetric keys depending on how they are used
- Typically long-term keys and ephemeral keys are used in establishment of session keys

Adversary capabilities

When analysing the security of key establishment protocol we assume an adversary that knows the details of the cryptographic algorithms involved (why?) and is able to:

- eavesdrop on all messages sent in a protocol
- alter all messages sent in a protocol using any information available
- re-route any message (including fabricated messages) to any user
- **obtain the value of the session key K_{AB} used in any previous run of the protocol**

Security goals

The security of key establishment protocols is defined by two properties

authentication: if a party A completes the protocol and believes (is convinced) that the session key K_{AB} is shared with party B, then K_{AB} should not be shared with a different party C

confidentiality: an adversary is unable to obtain the session key accepted by a particular party

- Authentication can be *mutual* (both parties achieve it) or *unilateral* (only provided on one side)
- Many real-world key establishment protocols achieve only unilateral authentication

Categories of key establishment protocols

- Three common approaches are:
  1. key pre-distribution where keys are set in advance
  2. key transport where one party chooses the key and distributes it
  3. key agreement where two or more parties contribute to the session key
- Each of these approaches can use symmetric-key cryptography or public-key cryptography

Key pre-distribution (pre-shared keys)

- A trusted authority (TA) generates and distributes long-term keys to all users when they join the system
- Simplest scheme assigns a secret key for each pair of users
    - But then the number of keys grows quadratically resulting in poor scalability
- TA only operates in the pre-distribution phase and need not be online afterwards

Session key transport with an online server

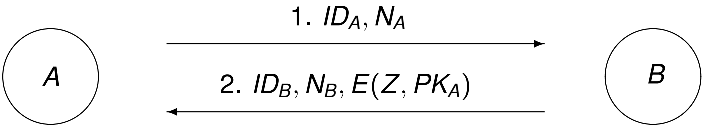
- TA shares a long-term shared key with each user
- TA generates and sends session keys to users when requested and protected by the long-term keys
- TA must be trusted and is a single point of attack
- Scalability can be a problem
- Kerberos (next few slides) is an example of this type of key establishment

- Typical advantages of using an online server are:
  - No need for computationally expensive public key algorithms.
  - Only a small amount of secure storage is required by each user to store the long-term key.
  - No certificate management overheads (distribution, validation, etc).
- Typical disadvantages include:
  - Requires that the TA be available online.
  - The TA is highly trusted and is a single point of attack. The security of the whole network depends on it.
  - Scalability can be a problem.

Key transport with asymmetric cryptography

- One user chooses key material and sends it encrypted with the other party's public key
- Each party can include a random *nonce* value to ensure that the key is new
- A key derivation function (KDF) binds the secret key material with other protocol elements to prevent certain attacks
- A standard KDF uses HMAC — think of a KDF as a hash function
- TLS up to version 1.2 includes this type of key establishment

Key transport protocol



- PK_A is A's public encryption key
- Z is a random value generated by B
- The session key can be

$$K_{AB} = KDF(Z, ID_A, ID_B, N_A, N_B)$$

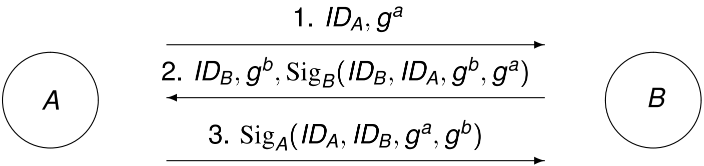
Key agreement

- Two parties each provide input to the keying material
- Usually add authentication with public keys, for example by signing the exchanged messages
- Diffie–Hellman protocol is a widely used key agreement protocol
- TLS includes Diffie–Hellman which is today the usual method of key establishment in TLS

Signed Diffie-Hellman

- A and B are two parties with identities ID_A, ID_B, who want to share a session key
- Computation takes place in a group G with generator g
- a, b are random values chosen by A and B in the range up to the order of G
- $\mathrm{Sig}_A(m)$ is a digital signature on message m by A
- $\mathrm{Sig}_B(m)$ is a digital signature on message m by B
- Both parties need each other's public signature verification keys

Signed Diffie–Hellman protocol



1. ID_A, g^a

2. $ID_B, g^b, \mathrm{Sig}_B(ID_B, ID_A, g^b, g^a)$

3. $\mathrm{Sig}_A(ID_A, ID_B, g^a, g^b)$

- A checks the signature received in step 2 and if it is valid A computes the shared secret: $Z = (g^b)^a = g^{ab}$
- Similarly B checks the signature received in step 3 and, if valid, computes the shared secret: $Z = (g^a)^b = g^{ab}$
- The session key can be

$$K_{AB} = KDF(Z, ID_A, ID_B, g^a, g^b)$$

Forward secrecy

- What happens when a long-term key is compromised?
- The attacker can now act as the owner of the long-term key
- Previous session keys may also be compromised

Forward secrecy definition

A key agreement protocol provides *(perfect) forward secrecy* if compromise of long-term private keys does not reveal session keys previously agreed using those long-term keys

- Remember that the attacker can see (and record) protocol messages
- Some use the term *perfect forward secrecy* while others simply say *forward secrecy* for the same thing

Post-compromise security (PCS)

- Sometimes we can recover after a long-term key is compromised – also known as *self-healing* protocols
- Only works when attacker is passive
- The long-term key must evolve over time so that the attacker becomes locked out with long-term key updates
- A way to do this is to send a new Diffie–Hellman share with every message and change the session key also after every message
- Used today in secure messaging in protocols such as Signal

Forward secrecy examples

- The protocol on slide 15 does **not** provide forward secrecy
- The protocol on slide 18 **does** provide forward secrecy, because the long-term (signing) keys are only used for authentication
- Neither of these two protocols provides post-compromise security

Key establishment using public keys

Can be achieved in several ways.

- Public announcement – Alice can broadcast her public key at large
  - Anyone can forge this – by the time Alice realises, forger has done some damage
- Publicly available directory
  - More secure, but still has vulnerabilities. Single point of failure
- Public-key authority
  - Central autority maintains a dynamic directory. Still some drawbacks: user must appeal to the authority for every other user it wants to contact, so it can be a bottleneck
- Public-key certificates
  - What is used in practice. A certificate is: a public key, an identifier of the key owner, and all this is signed by a trusted third party

Digital certificates

- When using a public key to encrypt a message or to verify a digital signature, it is essential to be confident of the correct binding between a public key and its owner
- Normally this is achieved through use of *digital certificates* which contain the public key and owner identity, and usually other information such as signature algorithm and validity period
- The certificate is digitally signed by a party trusted by the certificate verifier, normally called a *certification authority* or CA
- Certificates play a central role in key management for public key infrastructures

Public key infrastructure (PKI)

- In the NIST Special Publication 800-57 a public key infrastructure is defined as:
  "A framework that is established to issue, maintain, and revoke public-key certificates"
- A number of different legal or business entities may be involved including:
  - registration authorities which manage identities
  - naming authorities which manage domain naming
  - certfication authorities (CAs)
- We focus on technical issues in CAs

Certification Authority (CA)

- Creates, issues, and revokes certificates for users and other CAs
- Have a Certification Practice Statement (CPS) covering issues such as:
  - checks performed before certificate issue
  - physical, personnel and procedural security controls for the CA
  - technical and key pair protection and management controls
  - certificate revocation management procedures
  - audit procedures for the CA
  - accreditation information
  - legal and privacy issues and liability limitations

X.509 standard

- Widely used standard allowing flexible *extensions*
- Originally an ITU standard, now available as RFC 5280
- Important fields in X.509 digital certificates are:
  - Version number
  - Serial number (set by the CA)
  - Signature algorithm identifier (Algorithm used for dig sigs)
  - Issuer (Name of the CA)
  - Subject (Name of entity to which certificate is issued)
  - Public key information
  - Validity period
  - Digital signature (of the certificate, signed by the CA)

Using a certificate

- Certificates are verified by checking that the CA signature is valid and that any conditions set in the certificate are correct
- In order to verify a certificate, the user of the certificate must have the correct public key of the CA (why?)
- Users may obtain certificates in different ways, including:
  - sent by owners during a protocol run
  - distributed with web browsers
  - in public directories
  - as part of a DNS record

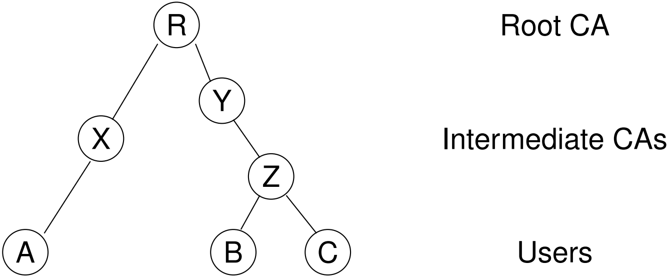
Certification paths

- If the public key of a CA, say $\mathtt{CA}_0$, is not already known and trusted, it can itself be certified by a different CA, say $\mathtt{CA}_1$
- In turn it is possible that the public key of $\mathtt{CA}_1$ is certified by $\mathtt{CA}_2$
- In this way a chain of trust can be set up, which is known as a *certification path*

$$\mathtt{CA}_n \to \mathtt{CA}_{n-1} \to \ldots \mathtt{CA}_1 \to \mathtt{CA}_0$$

- If an entity has a trusted copy of the public key of $\mathtt{CA}_n$, the chain of trust can be used with certificates for all the *intermediate CAs* to obtain a trusted copy of the public key of $\mathtt{CA}_0$

Hierarchical PKI



- CAs certify the public key of the entity below
- In a non-hierarchical PKI, certification can be done between any CAs: X can certify Y's public key or Z can certify Y's public key, for example

Browser PKI

- Multiple hierarchies with preloaded public keys as root CAs
- Other CAs and intermediate CAs can be added
- Your own certificates can also be added - you will be doing this in the lab
- Web servers send their public key and certificate to the browser at the start of a secure communication using the TLS protocol
- Root certificates are *self-signed* — the CA for the root CA is the root CA itself!

Revocation

- Sometimes it may be required to declare a certificate invalid even though its validity period is current
- In order to make this work the user must check to see which certificates have been revoked
- Two widely deployed mechanisms:
  1. Certificate revocation lists (CRL) – each CA periodically issues a list of revoked certificates which can be downloaded and then checked by clients
  2. Online certificate status protocol (OCSP) – a server maintains a current list of revoked certificates and responds to requests about specific certificates

Key distribution using symmetric key encryption

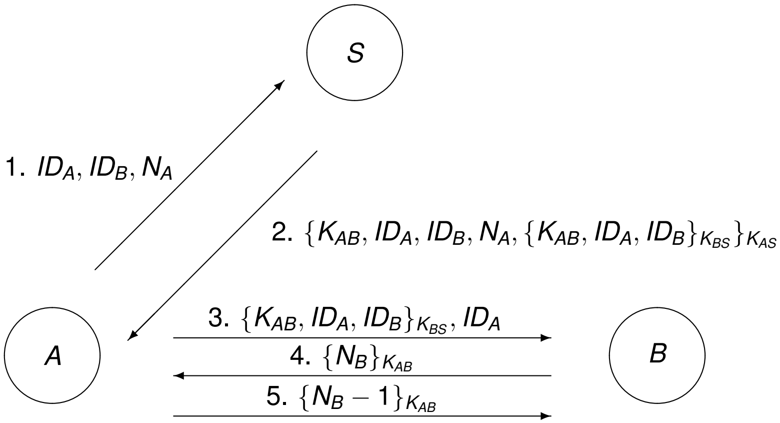
Options are as follows.

- Alice picks a key, physically delivers it to Bob
  - Need to meet physically
- A third party selects a key and physically delivers it to Bob
  - Need to meet physically
- Alice and Bob previously have shared a key – use that to update the key
  - But if an attacker gains access to the key at any point, it can decrypt all subsequent messages!
- Alice and Bob have an encrypted connection to (trusted third party) Charlie – Charlie can deliver new key to Alice and Bob

Needham–Schroeder protocol: notation

- Parties/Principals: A, B, S
  - A and B are two parties who want to establish a session key S is the trusted authority
- Shared Secret Keys: K_{AS}, K_{BS}, K_{AB}
  - A and S share long-term key K_{AS}
  - B and S share long-term key K_{BS}
  - K_{AB} is the new session key generated by S
- Nonces: N_A, N_B
  - Randomly-generated for one-time use (*n*-once)
- $S \to A : M$
  - S sends message M to A
- $\{X\}_K$
  - Authenticated encryption of message X using the shared secret key K

Needham–Schroeder protocol


S
1. ID_A, ID_B, N_A
2. $\{K_{AB}, ID_A, ID_B, N_A, \{K_{AB}, ID_A, ID_B\}_{K_{BS}}\}_{K_{AS}}$
3. $\{K_{AB}, ID_A, ID_B\}_{K_{BS}}, ID_A$
A
4. $\{N_B\}_{K_{AB}}$
B
5. $\{N_B - 1\}_{K_{AB}}$

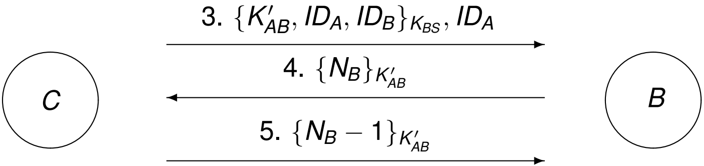
Needham–Schroeder protocol

One of the most widely known key establishment protocols

- Published by Needham and Schroeder in 1978
- The basis for many related protocols, including the Kerberos protocol we look at below
- The protocol is vulnerable to a *replay attack* found by Denning and Sacco in 1981
- An attacker is able to replay old protocol messages and the honest party accepts an old session key

Replay attack on Needham–Schroeder protocol

- Assume an attacker C obtains a session key K'_{AB} previously established between A and B
- In the attack, C masquerades as A and is thus able to persuade B to use the old key K'_{AB}

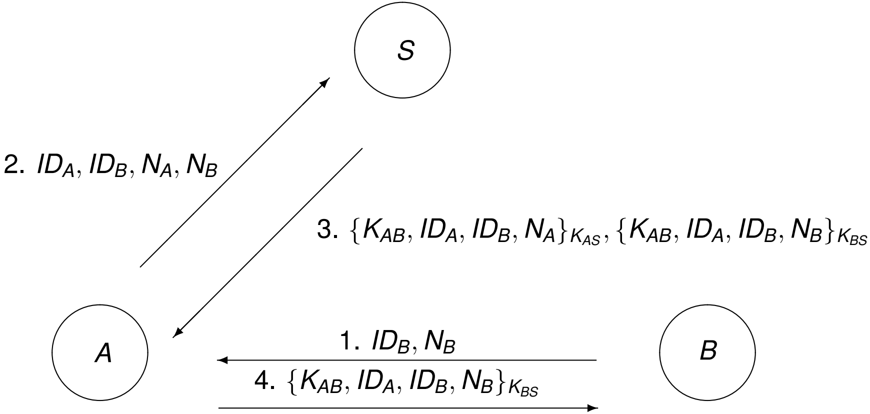


3. $\{K'_{AB}, ID_A, ID_B\}_{K_{BS}}, ID_A$

4. $\{N_B\}_{K'_{AB}}$

5. $\{N_B - 1\}_{K'_{AB}}$

Repaired Needham–Schroeder protocol



1. ID_B, N_B

2. ID_A, ID_B, N_A, N_B

3. $\{K_{AB}, ID_A, ID_B, N_A\}_{K_{AS}}, \{K_{AB}, ID_A, ID_B, N_B\}_{K_{BS}}$

4. $\{K_{AB}, ID_A, ID_B, N_B\}_{K_{BS}}$

Freshness

- To defend against replay attacks, it is critical that the key established be *fresh* (new) for each session
- There are three basic mechanisms that can be used to achieve freshness:
  1. random challenges (nonces)
  2. timestamps (a string on the current time)
  3. counters (increased for each new message)
- The repaired Needham–Schroeder protocol above uses random challenges to provide freshness
- Timestamps or counters can also be used

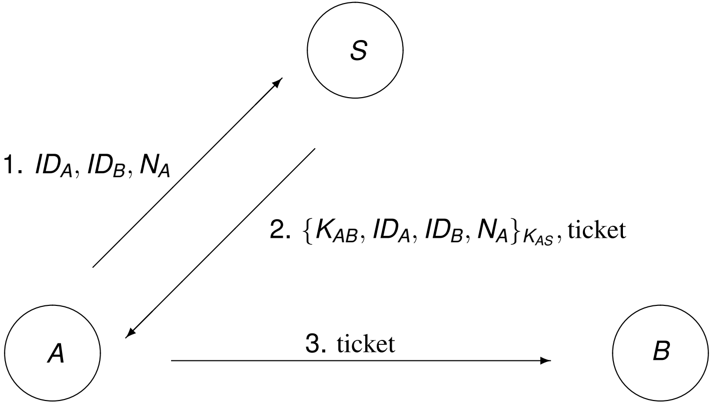
Tickets

- An alternative way to fix the Needham–Schroeder protocol uses a key with a validity period
- Suppose entity *A* wishes to obtain access to the server *B*
- The authentication server *S* issues a *ticket* to allow *A* to obtain access
- A ticket has the format

  $$\{K_{AB}, ID_A, ID_B, T_B\}_{K_{BS}}$$

  where T_B is a timestamp which we can also interpret as a validity period
- *A* can obtain the ticket and use it to gain access to *B* at any time while T_B is still valid

Repaired Needham–Schroeder using tickets



Kerberos

- Originally developed in early 1980s, latest version Kerberos V5 released in 1993 described in RFC 4120 (2005)
- Since Windows 2000, Kerberos V5 is the default Windows domain authentication method
- A single sign-on (SSO) solution: users only need to enter usernames and passwords once for a session
- Provide access selectively for a number of different online services using individual tickets
- Establish session key to deliver confidentiality and integrity services for each service access

Kerberos three level protocol

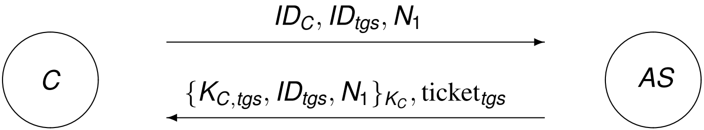
Level 1 Client *C* interacts with authentication server *AS* in order to obtain a ticket-granting ticket – happens once for a session (maybe a working day)

Level 2 Client *C* interacts with ticket-granting server *TGS* in order to obtain a service ticket – happens once for each server during the session

Level 3 Client *C* interacts with application server *V* in order to obtain a service – happens each time the client requires service during the session

- The protocol descriptions following have some simplifications

Level 1: Interaction with authentication server



C

ID_C, ID_{tgs}, N_1

$\{K_{C,tgs}, ID_{tgs}, N_1\}_{K_C}, \mathrm{ticket}_{tgs}$

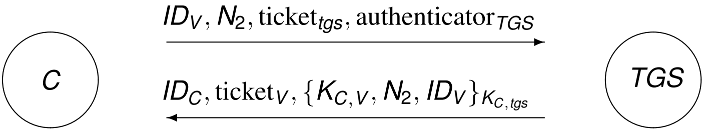
AS

- $\mathrm{ticket}_{tgs} = \{K_{C,tgs}, ID_C, T_1\}_{K_{tgs}}$ for some validity period T_1
- Result: user has ticket-granting ticket which can be used to obtain different service-granting tickets

Level 1: notes

- K_C is the symmetric key shared with the authentication server AS, typically it is generated by the workstation of C from a password entered by C at login time
- $K_{C,tgs}$ is a new symmetric key generated by AS to share with the ticket granting server TGS
- N_1 is a nonce used by C to check that the key $K_{C,tgs}$ is fresh
- K_{tgs} is a long-term key shared between AS and TGS

Level 2: Interaction with ticket-granting server



C

$ID_V, N_2, \mathrm{ticket}_{tgs}, \mathrm{authenticator}_{TGS}$

$ID_C, \mathrm{ticket}_V, \{K_{C,V}, N_2, ID_V\}_{K_{C,tgs}}$

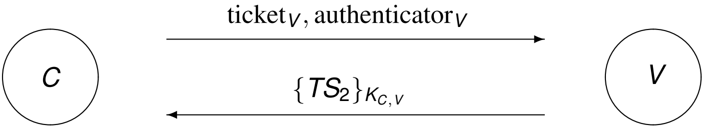
TGS

- $\mathrm{ticket}_V = \{K_{C,V}, ID_C, T_2\}_{K_V}$ for some validity period T_2
- $\mathrm{authenticator}_{TGS} = \{ID_C, TS_1\}_{K_{C,tgs}}$ for some timestamp TS_1
- Result: user has service-granting ticket which can be used to obtain access to a specific server

Level 2: notes

- The ticket_{tgs} is the same as sent in the level 1 interaction
- $K_{C,V}$ is the session key to be used with server V
- N_2 is a nonce used by C to check that the key $K_{C,V}$ is fresh.
- TGS first obtains $K_{C,tgs}$ from ticket_{tgs} and then checks the fields in the authenticator are valid – includes checking that TS_1 is recent and that C is authorized to access service V
- In practice both AS and TGS may be the same machine

Level 3: Interaction with application server



- $\text{authenticator}_V = \{ID_C, TS_2\}_{K_{C,V}}$ for some timestamp TS_2
- Result: user has secure access to a specific server V

Level 3: notes

- The ticket_V is the same as sent in the level 2 interaction
- $K_{C,V}$ is contained in ticket_V and was also sent to C in the level 2 interaction
- The reply from V is intended to provide mutual authentication so that C can check it is using the right service V

Kerberos limitations

- Limited scalability: even though different realms are supported, each realm needs to share a key with each other realm
- Suited for corporate environments with shared trust (although public key variants exist)
- Offline password guessing is a possible attack when the client key K_C is derived from a human memorable password
- Kerberos standard does not specify how to use the session key once it is established

Lecture 13: The Transport Layer Security (TLS) Protocol

TTM4135

Relates to Stallings Chapter 17

Spring Semester, 2025

Motivation

- Transport Layer Security (TLS) is a cryptographic protocol designed to provide communication security over a computer network
- TLS is probably the most widely used security protocol in use today in the real world
- TLS is used to secure communications with banks, online shops, email providers and much more
- TLS uses most of the mainstream cryptographic algorithms which we have studied in this course
- TLS is a very complex protocol and has been the subject of many attacks, and subsequent repairs

Outline

History and Overview

TLS Record Protocol

TLS Handshake Protocol

Attacks on TLS

SSL/TLS history

- 1994: Netscape Communications developed Secure Sockets Layer (SSL) 2.0. Should no longer be used.
- 1995: Netscape release SSL 3.0. Should no longer be used.
- RFC 2246 issued in 1999 by IETF documenting Transport Layer Security (TLS) protocol 1.0, similar to SSL 3.0.
- TLS 1.1 specified in 2006 in RFC 4346. Fixes some problems with non-random IVs and exploitation of padding error messages.
- TLS 1.2 specified in 2008 in RFC 5246. Allows use of standard authenticated encryption (instead of separate encryption and MAC).

TLS 1.2 vs. TLS 1.3

- TLS 1.3 standardised in 2018 in RFC 8446 with significant differences from TLS 1.2
- TLS 1.2 is still the most widely supported version today so we focus on **TLS 1.2 in this lecture**
- We focus on TLS 1.3 in the next lecture and explain the main differences there

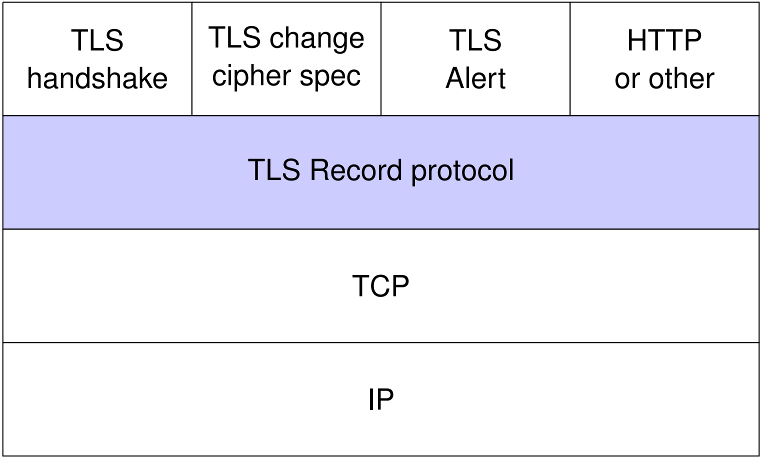
TLS uses

- TLS is a cryptographic services protocol based upon PKI and commonly used on the Internet
- Often used to allow browsers to establish secure sessions with web servers
- Many other application areas

TLS: Architecture overview

- TLS is not a single protocol but rather two layers of protcols (see next slide)
- Consists of 3 higher level protocols:
  - TLS handshake protocol to set up sessions
  - TLS alert protocol to signal events such as failures
  - TLS change cipher spec protocol to change the cryptographic algorithms
- The TLS record protocol provides basic services to various higher level protocols

TLS: Protocol stack



TLS handshake

TLS change cipher spec

TLS Alert

HTTP or other

TLS Record protocol

TCP

IP

TLS Alert and TLS change cipher spec protocols

- The Alert protocol handles connections, by sending an "alert" message of various degrees of severity
- Three types of alerts:
  - warning alerts
  - `close_notify` alerts
  - fatal alerts
- Improper handling of alert messages can lead to truncation attacks
- The change cipher spec protocol is normally used after the handshake protocol to indicate commencement of secure traffic

TLS ciphersuites

- TLS ciphersuites specify the public key algorithms used in the handshake protocol and the symmetric algorithms used in the record protocol
- Over 300 standardized ciphersuites, many of which are weak and should no longer be used
- Full list is held by IANA
- An example ciphersuite, mandatory in TLS 1.0 and 1.1, is TLS_RSA_WITH_3DES_EDE_CBC_SHA. This means that:
  - the key exchange will use RSA to encrypt a secret chosen by the client (see following slides);
  - triple DES (Encrypt-Decrypt-Encrypt) in CBC mode will be used for encryption;
  - SHA-1 will be used for the HMAC for data integrity.

HMAC – reminder

- Proposed by Bellare, Canetti, Krawczyk in 1996
- Built from any iterated cryptographic hash function H, e.g., MD5, SHA-1, SHA-256, …
- Standardized and used in many applications including TLS and IPsec
- Details in FIPS-PUB 198-1 (July 2008)

HMAC construction – reminder

Let H be any iterated cryptographic hash function. Then define:

$$\mathrm{HMAC}(M, K) = H\left((K \oplus \mathrm{opad}) \parallel H((K \oplus \mathrm{ipad}) \parallel M) \right)$$

where

- M: message to be authenticated
- K: key padded with zeros to be the block size of H
- opad: fixed string `0x5c5c5c...5c`
- ipad: fixed string `0x363636...36`
- $\parallel$ denotes concatenation of bit strings.
- HMAC is secure (unforgeable) if H is collision resistant or if H is a pseudorandom function.

HMAC – reminder

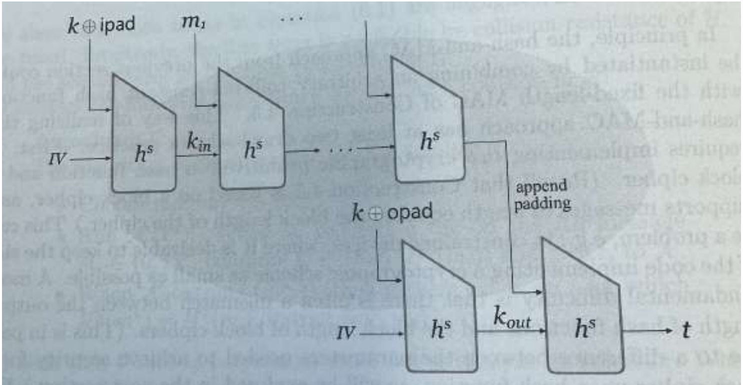


FIGURE 6.2: HMAC, pictorially.

Common TLS 1.2 ciphersuites

- Handshake algorithms (all use signed Diffie–Hellman)
  - DHE-RSA Ephemeral Diffie-Hellman with RSA signatures
  - ECDHE-RSA Elliptic curve DHE with RSA signatures
  - DHE-DSS DHE with DSS signatures
- Record algorithms
  - AES-GCM AES authenticated encryption with GCM mode
  - AES-CBC-SHA256 AES in CBC mode with HMAC from SHA256
  - CHACHA20-POLY1305 ChaCha stream cipher with Poly1305 MAC

Record protocol overview

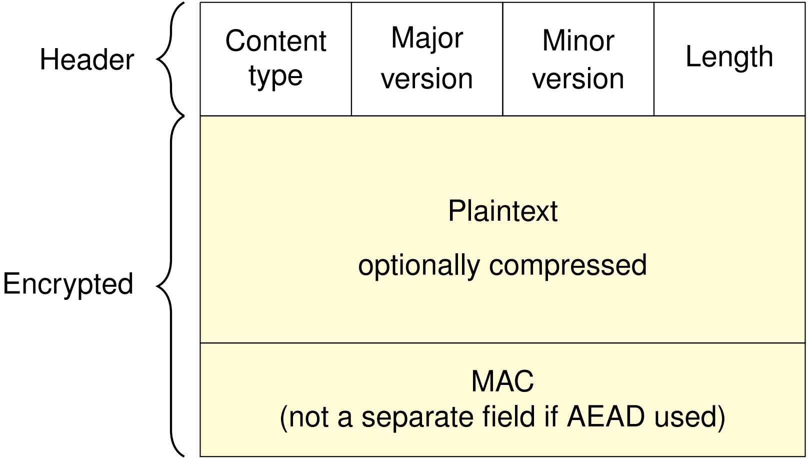
Provides two services for higher-layer protocols.

Message confidentiality: Ensure that the message contents cannot be read in transit

Message integrity: Ensure that the receiver can detect if a message is modified in transit

- These services can be provided by a symmetric encryption algorithm and a MAC
- From TLS 1.2, these services are often provided with authenticated encryption with associated data (AEAD) modes CCM or GCM
- The handshake protocol establishes symmetric keys (session keys) to use with these mechanisms

Record protocol format



Header

Content type | Major version | Minor version | Length

Encrypted

Plaintext

optionally compressed

MAC
(not a separate field if AEAD used)

TLS: Record protocol header

- Content Type: Defined content types are:
  - change-cipher-spec
  - alert
  - handshake
  - application-data
- Protocol Version
  - Major Version: 3 for TLS
  - Minor Version: 1 for TLS v1.0; 2 for TLS v1.1; 3 for TLS v1.2.
- Length: length in octets of the data

Record protocol operation

- Fragmentation: Each application layer message is fragmented into blocks of 2^{14} bytes or less
- Compression: Optionally applied – default compression algorithm is null
- Authenticated data: consists of the (compressed) data, the header, and an implicit record sequence number
- Plaintext: Compressed data and the MAC, if present
- Session keys for the MAC and encryption algorithms, or AEAD algorithm, are established during the handshake protocol
- The encryption and MAC algorithms are specified in the negotiated *ciphersuite*

Record protocol cryptographic algorithms

MAC: The algorithm used is HMAC in all TLS versions using a negotiated hash function. SHA-2 is allowed only from TLS 1.2.

Encryption: Either a negotiated block cipher in CBC mode or a stream cipher. Most common block ciphers are AES and 3DES. RC4 originally supported in TLS 1.2. For block ciphers, padding is applied after the MAC to make a multiple of the cipher block size.

AEAD: Allowed instead of encryption and MAC in TLS 1.2. Usually AES in CCM or GCM modes. Authenticated additional data is the header and implicit record sequence number.

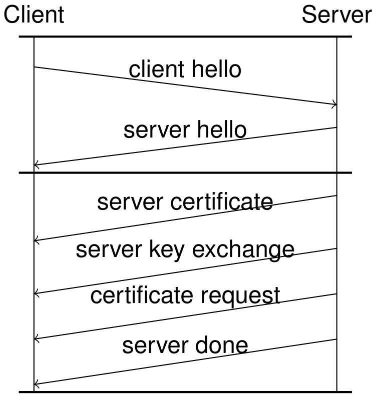
Handshake protocol purpose

- Negotiates the version of TLS and the cryptographic algorithms to be used
- Establishes a shared session key for use in the record protocol
- Authenticates the server
- Authenticates the client (optional)
- Completes the session establishment
- Several variations of the handshake:
  - RSA variant (still supported but not recommended)
  - Diffie-Hellman variant (recommended)
  - Pre-shared key variant
  - Mutual authentication or server-only authentication

Handshake protocol - four phases

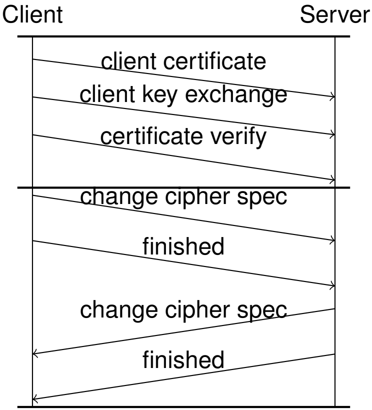
- Phase 1: Initiates the logical connection and establishes its security capabilities
- Phases 2 and 3: Performs key exchange with messages and message content depending on the handshake variant negotiated in phase 1
- Phase 4: Completes the setting up of a secure connection

Handshake protocol - phases 1 and 2



- Phase 1: Client and server negotiate version, cipher suite and compression and exchange nonces
- Phase 2: Server sends certificate and key exchange message (if needed)

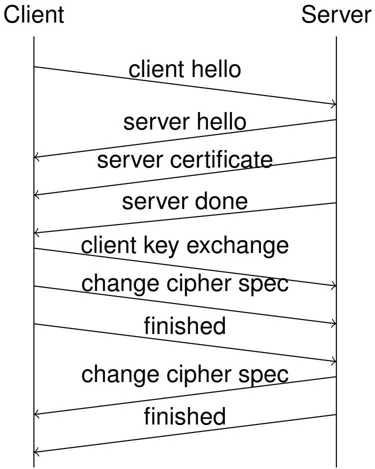
Handshake protocol - phases 3 and 4



- Phase 3: Client sends certificate and key exchange message
- Phase 4: Client and server start secure communications

Finished messages include a check value (pseudo-random function) of all the previous messages

RSA-based handshake protocol



- Simplest variation with server-only authentication and server public key suitable for RSA encryption.
- On completion of Phase 1, assume that RSA-based key exchange has been selected.

Main handshake messages

Client hello States highest version of TLS available, advertises ciphersuites available to the client and sends client nonce N_C

Server hello Returns the selected version and ciphersuite and sends server nonce N_S

Server key exchange Server's input to key exchange

Client key exchange Client input to key exchange

Change-cipher-spec Switch to newly negotiated ciphersuite for record layer

Ephemeral Diffie–Hellman handshake variant

Server key exchange Diffie–Hellman generator and group parameters and server ephemeral Diffie-Hellman value, all signed by server

Client key exchange Client ephemeral Diffie-Hellman value. This is optionally signed by the client if the client certificate is used

- Pre-master secret *pms* is the shared Diffie–Hellman secret
- Provides forward secrecy and therefore *recommended* today

RSA handshake variant

Server key exchange Not used

Client key exchange Key transport of pre-master secret *pms*

- Client selects a random *pre-master secret*, *pms*
- Client encrypts *pms* with the server's public key and sends the ciphertext to the server
- Server decrypts using its private key to recover *pms*

No forward secrecy and *not recommended* today

Generating session keys

- The master secret, *ms*, is defined using the premaster secret *pms*, as:

$$ms = PRF(pms, \text{“master secret”}, N_C \parallel N_S)$$

- As much keying material is generated as is required by the ciphersuite using:

$$k = PRF(ms, \text{“key expansion”}, N_S \parallel N_C)$$

- Independent session keys are partitioned from *k* in each direction (a write key and a read key on each side)
- Depending on ciphersuite, keying material may include:
  - encryption key
  - MAC key
  - IV

The pseudorandom function PRF

- The PRF (pseudo-random function) is built from HMAC with a specified hash function
  - The PRF in TLS 1.0 and TLS 1.1 is based on a combination of MD5 and SHA-1
  - In TLS 1.2 the PRF is based on SHA-2
- For example, in TLS 1.2:

$$\begin{aligned} PRF(K, label, r) &= HMAC(K, A(1) \parallel label \parallel r) \parallel \\ &\quad HMAC(K, A(2) \parallel label \parallel r) \parallel \\ &\quad \ldots \end{aligned}$$

where $A(0) = r$, $A(i) = HMAC(K, A(i-1))$ and HMAC uses a specified SHA-2 variant, typically SHA256, as its hash function, r is a seed.

Other handshake variants

Diffie-Hellman (DH) The parties use **static** Diffie–Hellman with certified keys — if the client does not have a certificate (usual on the Internet) then the client uses an ephemeral Diffie-Hellman key

Anonymous Diffie-Hellman (DH_Anon) The ephemeral Diffie-Hellman keys are not signed at all, so only protects against passive eavesdropping

The above methods for the handshake protocol are possible but not recommended today

Forward secrecy

- Forward secrecy is the property that compromise of long-term keys should not lead to compromise of session keys established before the long-term key compromise took place.
- A typical way to achieve forward secrecy is to use Diffie–Hellman key exchange with the exchange authenticated using signatures from the long-term keys.
- Using the RSA-based handshake in TLS does not achieve forward secrecy but is currently still used in many ciphersuites.
- Several ciphersuites using Diffie–Hellman, including elliptic curve Diffie–Hellman, are defined for TLS which provide forward secrecy.

SSL and TLS Limitations

- Higher layers should not be overly reliant on SSL or TLS always negotiating the strongest possible connection between two peers
- There are a number of ways a man-in-the-middle attacker can attempt to make two entities drop down to the least secure method they support.
  - Known as a downgrade attack.
- For example, an attacker could block access to the port a secure service runs on, or attempt to get the peers to negotiate an unauthenticated connection.

TLS protocol summary

- TLS includes two main protocols: the Handshake Protocol and the Record Layer Protocol
- New versions have been rolled out as understanding of cryptography and potential attacks increase
- Backward compatibility is a problem:
  - SSL 3.0 finally deprecated in 2015
  - TLS 1.0 more than 20 years old and still supported widely
- TLS assumes reliable delivery of messages, provided by TCP

TLS limitations

- In the past few years there have been many practical attacks on TLS
- Many servers do not support the latest versions of TLS and/or have not protected against known attacks
- SSL Pulse survey gives an up-to-date picture of current attacks
- Good coverage of some attacks is given on Matt Green's blog
- A selection of some previous attacks is on the following slides

The BEAST attack

- BEAST (Browser Exploit Against SSL/TLS) exploits non-standard use of IV in CBC mode encryption - IVs are chained from previous ciphertexts
- Allows attacker to recover plaintext byte by byte
- Known as a theoretical weakness since 2002, but only demonstrated in 2011
- From TLS 1.1 only random IV is allowed
- Most browsers now implement a mitigation strategy based on splitting plaintext into first byte + remainder to force a randomised IV including a MAC computation
- No longer considered a realistic threat

The CRIME and BREACH attacks

- Side channel attacks based on **compression** - different inputs result in different amounts of compression
- CRIME (Compression Ratio Info-leak Made Easy) exploits compression in TLS, while BREACH (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext) exploits compression in HTTP
- Idea of attack known already in 2002 but practically demonstrated in 2012
- Commonly recommended to switch off compression in TLS but switching off in HTTP too results in big performance penalty

Padding oracles and the POODLE attack

- A padding oracle is a way for an attacker to know if a message in a ciphertext was correctly padded
- In 2002 it was shown how in theory CBC mode encryption can provide a padding oracle due to its error propagation properties
- Applied to TLS in a variety of attacks
- Main mitigation is a uniform error response, so that the attacker cannot distinguish padding errors from MAC errors
- POODLE (Padding Oracle On Downgraded Legacy Encryption) published in October 2014 forces downgrade to SSL 3.0 and then runs padding oracle attack

The Heartbleed bug

- An implementation error in OpenSSL
- Based on missing bounds check in *heartbeat* messages
- Allows memory leakage from server which is likely to include session keys and long-term keys
- Discovered April 2014 and required updating of many server keys after bug was fixed
- Is it reasonable that big companies use free software for securing important transactions?

TLS Timing (Padding) Oracle Attack

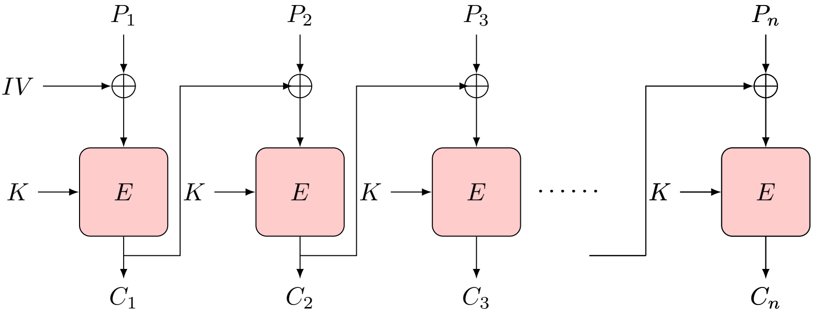
- Attack discovered by Nadhem AlFardan and Kenny Patterson from (formerly) Royal Holloway, University of London, UK (now at ETH)
- Dubbed “Lucky 13”
- TL;DR
    - *“There is a subtle timing bug in the way that TLS data decryption works when using the (standard) CBC mode ciphersuite. Given the right set of circumstances, an attacker can use this to completely decrypt sensitive information, such as passwords and cookies. ”*
    - Borderline practical in DTLS, more on the theoretical side if using TLS

https://blog.cryptographyengineering.com/2013/02/04/attack-of-week-tls-timing-oracles/
https://en.wikipedia.org/wiki/Lucky_Thirteen_attack

Cipher Block Chaining (CBC) mode – reminder

- CBC “chains” the blocks together
- A random initialisation vector *IV* is chosen and sent together with the ciphertext blocks
- *Encryption*:
  - $C_t = E(P_t \oplus C_{t-1}, K)$, where $C_0 = IV$
  - P_t is XOR’d with the previous ciphertext block C_{t-1}, and encrypted with key K to produce ciphertext block C_t
  - IV is used for the value C_0 and sent with $C_1, \ldots C_n$
- *Decryption*:
  - $P_t = D(C_t, K) \oplus C_{t-1}$, where $C_0 = IV$
  - C_t is decrypted with the key K, and XOR’d with the previous ciphertext block C_{t-1} to produce plaintext block P_t
  - As in encryption, C_0 is used as the IV

CBC mode encryption



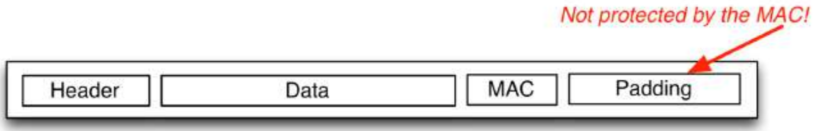
- IV and blocks $C_1, C_2, \ldots C_n$ are sent

The attack

- Recall that we saw that you should always encrypt a message first, then apply the MAC to the resulting ciphertext
- But TLS gets this backwards
- Upon encrypting a record,
  - The sender first applies a MAC to the plaintext
  - Then adds up to 255 bytes of padding to get the message up to a multiple of the cipher (e.g., AES) block size
  - Only then does it CBC-encrypt the record.

The attack

The critical point is that the padding is *not* protected by the MAC.



This means an attacker can tamper with the padding by flipping specific bits in the ciphertext, leading to a *padding oracle attack*. She can re-transmit to the server for decryption. If the attacker can learn whether her changes affected the padding, she can use this information to adaptively decrypt the whole record.

The attack

- These kinds of attacks were known to TLS designers
- But, instead of fixing, they patched it
- They did so by eliminating eliminate any error messages that could indicate whether the padding check failed
- But researchers realised they could run a *timing* attack instead
  - Time how long decryption took, deduce whether there was a padding failure or not
  - Implementations would first check padding, then return immediately (without checking the MAC) if this failed

The attack

“[T]he best way to do this is to compute the MAC even if the padding is incorrect, and only then reject the packet. For instance, if the pad appears to be incorrect, the implementation might assume a zero-length pad and then compute the MAC." (TLS 1.2 spec)

- When the padding check fails, the decryptor doesn’t know how much padding to strip off
- Meaning doesn’t know how much data to MAC
- Recommended countermeasure is to assume no padding, then MAC the whole blob
    - The results is that the MAC computation can take longer when the padding is damaged

The attack

“This leaves a small timing channel, since MAC performance depends to some extent on the size of the data fragment, but it is not believed to be large enough to be exploitable, due to the large block size of existing MACs and the small size of the timing signal.”

- This remained true indeed for several years
- The attack showed that it *was* possible to distinguish, at least from a relatively short distance
- Partly due to advances in computer hardware
- The new technique was able to measure timing differentials of less than 1 microsecond over a LAN connection

TLS security summary

- Different kinds of attacks: implementation errors, poor choice of cryptographic primitives, flaws in protocol.
- Backward compatibility is a problem (Downgrade attacks).
- Several examples of the principle that “attacks only get better” over time.
- Complexity is a major problem. TLS 1.3 will remove many options, both in cipher suites and protocol options.
- TLS 1.3 will simplify the handshake
- TLS 1.3 adds new features (e.g. 0-RTT mode) which present new challenges.

Lecture 14: TLS 1.3 and IPsec

TTM4135

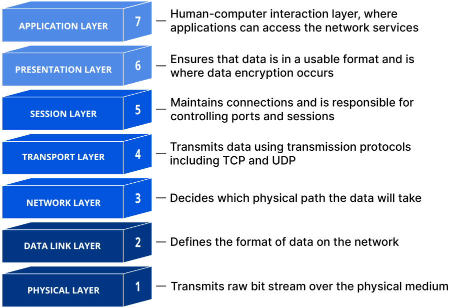
Relates to Stallings Chapters 17 and 20

Spring Semester, 2025

Motivation

- TLS 1.3 is the latest version of the Transport Layer Security protocol and has significant changes from earlier versions affecting both security and efficency
- Internet Protocol security (IPsec) is a framework for ensuring secure communications over Internet Protocol (IP) networks
- IPsec provides similar security services as TLS, but at a lower layer in the communications protocol stack

OSI model – TLS/ IPsec


APPLICATION LAYER
7
Human-computer interaction layer, where applications can access the network services
PRESENTATION LAYER
6
Ensures that data is in a usable format and is where data encryption occurs
SESSION LAYER
5
Maintains connections and is responsible for controlling ports and sessions
TRANSPORT LAYER
4
Transmits data using transmission protocols including TCP and UDP
NETWORK LAYER
3
Decides which physical path the data will take
DATA LINK LAYER
2
Defines the format of data on the network
PHYSICAL LAYER
1
Transmits raw bit stream over the physical medium

TLS operates at the application layer, IPsec operates at the network layer.

Outline

TLS 1.3
- TLS 1.3 Development
- TLS 1.3 Differences

IP Layer Security (IPsec)
- IPsec Architectures
- IPsec Protocols
- IPsec Modes

Why was TLS 1.3 needed?

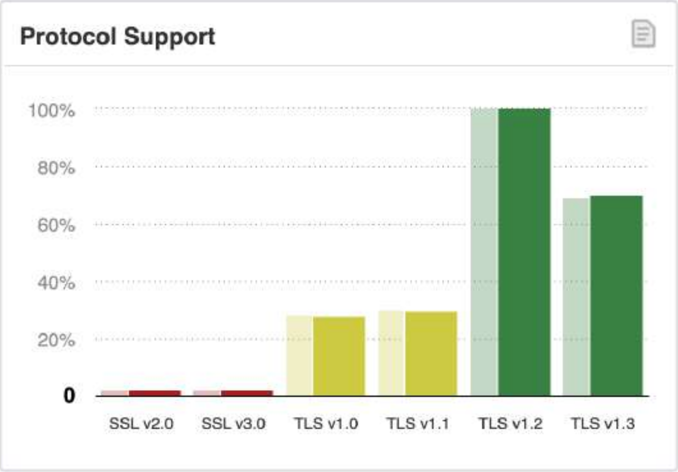
Efficiency: In earlier TLS versions need at least two round trip times (RTT) before data can be sent

Security: Many security problems in earlier TLS versions

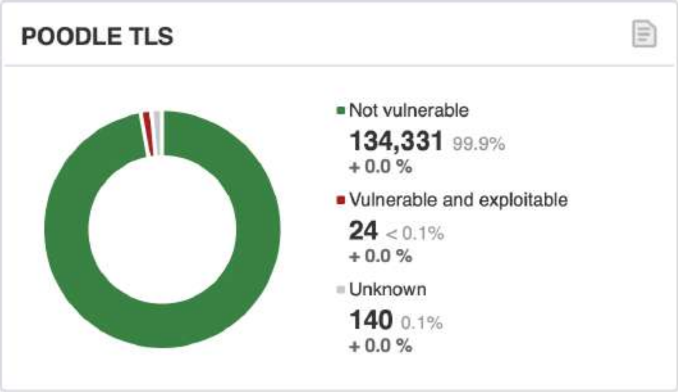
- Protocol was too complex
- Protocol supported old and weak ciphersuites

- New protocol designed to support sound cryptographic principles and aims to achieve *provable security*
- First drafted in 2014 in close cooperation between academics, practitioner community and developers
- Internet proposed standard RFC 8446 January 2018
- Today supported in around 66% of popular web servers according to SSL Pulse alongside earlier TLS versions

Protocol support



Poodle



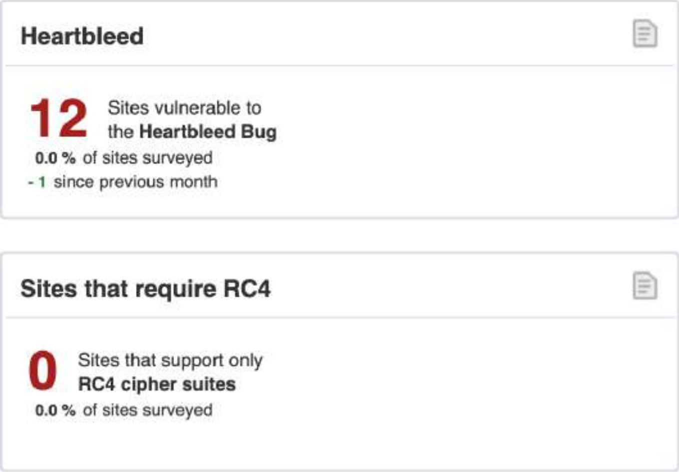
POODLE TLS

🟩 Not vulnerable
134,331 99.9%
+ 0.0 %

🟥 Vulnerable and exploitable
24 < 0.1%
+ 0.0 %

⬜ Unknown
140 0.1%
+ 0.0 %

Heartbleed – RC4



Heartbleed

12 Sites vulnerable to the **Heartbleed Bug**

0.0 % of sites surveyed

- 1 since previous month

Sites that require RC4

0 Sites that support only **RC4 cipher suites**

0.0 % of sites surveyed

Some concrete changes from TLS 1.2 to TLS 1.3

Some items removed:

- static RSA and DH key exchange (why?)
- renegotiation
- SSL 3.0 negotiation
- DSA in finite fields
- data compression
- non-AEAD cipher suites

Some items added:

- zero round-trip time (0-RTT) mode from pre-shared keys
- post-handshake client authentication through “certificate verify” signature
- more AEAD ciphersuites

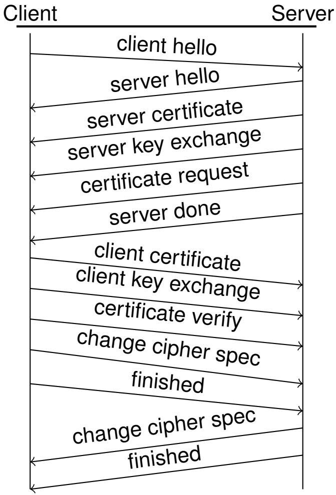
TLS 1.3 handshake protocol: hello messages

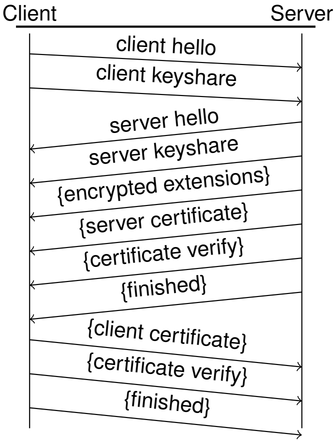
- Client sends keyshare field in client hello for one or more anticipated ciphersuites
- Server can obtain session key on receipt of client hello if:
  - server accepts one the clients ciphersuites
  - client keyshare matches the accepted ciphersuite
- If the above conditions fail then:
  - server sends an optional Hello Retry Request
  - client responds if there is an acceptable alternative ciphersuite
- Usually this results in saving a *whole round trip* of communication

TLS 1.3 handshake protocol: other messages

- Only client and server hello/keyshare messages are not cryptographically protected — all later parts of the protocol use handshake traffic keys
- Key calculation now uses the standard HKDF (hash key derivation function) to derive the individual keys instead of the ad hoc PRF used in TLS 1.2
- Several different key types derived from master secret:
  - *handshake traffic keys* to protect handshake protocol
  - *application traffic keys* for client-server traffic
  - *early data keys* for 0-RTT data (see below)
- Various “tricks” used to allow interoperability with devices that only accept earlier TLS versions

Handshake: TLS 1.2 (left) to TLS 1.3 (right)





{} protected by handshake traffic keys

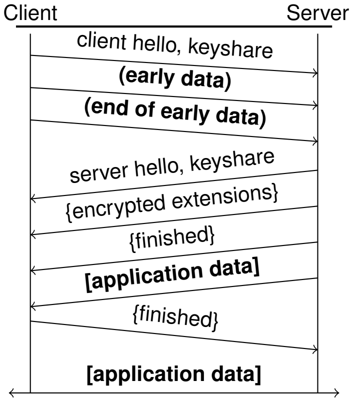
Client authentication

- In TLS 1.2 and 1.3 it is optional for the client to send a certificate and authenticate using a `CertificateVerify` message
- The `CertificateVerify` message includes a signature (with the secret key corresponding to the public key in the certificate) which can be verified using the public key in the certificate
- TLS 1.3 adds a *post-handshake client authentication* extension; if this is used then the server may request client authentication *at any time* after the handshake completed
- The client responds with its certificate and a signature in the form of `CertificateVerify`

0-RTT in TLS 1.3

- In a 0-RTT key establishment parties can start sending application data immediately, so-called early data
- 0-RTT in TLS 1.3 is based on a pre-shared key (PSK)
- PSK can either be agreed outside TLS or from an earlier TLS session
- At the end of the handshake protocol the server can send to the client one or more *new session tickets* as PSKs
- A client may start a new PSK session without negotiating version and ciphersuite

0-RTT in TLS 1.3



() protected by early data keys
{} protected by handshake traffic keys
[] protected by further traffic keys

- Only possible with pre-shared key
- Pre-shared key is used to authenticate DH
- Early data is optional and lacks forward secrecy

TLS 1.3 ciphersuites

- Handshake always uses Diffie-Hellman option so ciphersuite specifies only:
  - which AEAD cipher to use in Record layer
  - hash function to use for KDF
- TLS 1.2 and lower ciphersuite values cannot be used with TLS 1.3 and *vice versa*
- Mandatory to implement ciphersuite: `TLS_AES_128_GCM_SHA256`
- Other recommended ciphersuites: `TLS_AES_256_GCM_SHA384`, `TLS_CHACHA20_POLY1305_SHA256`, `TLS_AES_128_CCM_SHA256`, `TLS_AES_128_CCM_8_SHA256`

ChaCha algorithm

- Stream cipher defined in RFC 8439 together with a message authentication code (MAC) called Poly1305
- Available in a TLS ciphersuite (RFC 7905)
- Designed by D. J. Bernstein in 2008
- Faster than AES, except for processors with AES hardware support (most modern desktop computers)
- Combines $\oplus$, addition modulo 2^{32} and rotation operations over 20 rounds to produce 512 bits of keystream. An example of an *add-rotate-xor* or ARX cipher.
- 256-bit key

TLS 1.3 main improvements

Efficiency
- Saving of one round trip time in handshake
- Can set up follow-on session with 0-RTT

Security
- Only forward-secret key exchange now allowed
- Many legacy cipher suites no longer allowed
- Renegotiation option removed
- Formal security proofs

Selfie Attack on TLS 1.3

- Published in March 2019 by Drucker and Gueron
- Breaks mutual authentication in PSK mode
- Suppose Alice shares a PSK with Bob
- Attacker reflects messages back to herself so client Alice believe she is talking to Bob while she is actually talking with server Alice
- Case is not covered in formal analysis of TLS 1.3
- Can be prevented by forbidding to share PSK between more than one server and one client

IPsec: Introduction

- Standardised in RFCs 4301-4304 (2005) with crypto algorithms updated in subsequent RFCs
- Provides protection for any higher layer protocol
- Uses encryption, authentication and key management algorithms
- Most commonly used to provide Virtual Private Networks (VPNs)
- Provides a security architecture for both IPv4 and IPv6

Security services

Message confidentiality Protects against unauthorised data disclosure by the use of encryption

Message integrity Detects if data has been changed by using a message authentication code (MAC) or authenticated encryption

Limited traffic analysis protection Eavesdropper on network traffic should not know which parties communicate, how often, or how much data is sent

Message replay protection The same data is not replayed and data is not delivered badly out of order

Peer authentication Each IPsec endpoint confirms the identity of the other IPsec endpoint

Gateway-to-gateway architecture

- Provides secure network communications between two networks
- Network traffic is routed through the IPsec connection, protecting it appropriately
- Only protects data between the two gateways
- Most often used when connecting two secured networks, such as linking a branch office to headquarters over the Internet
- Can be less costly than private wide area network (WAN) circuits

Host-to-gateway architecture

- Commonly used to provide secure remote access.
- The organization deploys a virtual private network (VPN) gateway onto their network
- Each remote access user establishes a VPN connection between the local computer (host) and the gateway
- VPN gateway may be a dedicated device or part of another network device
- Most often used when connecting hosts on unsecured networks to resources on secured networks

Host-to-host architecture

- Typically used for special purpose needs, such as system administrators performing remote management of a single server
- Only model that provides protection for data throughout its transit (end-to-end)
- Resource-intensive to implement and maintain in terms of user and host management
- All user systems and servers that will participate in VPNs need to have VPN software installed and/or configured
- Key management is often accomplished through a manual process

IPsec protocol types

Encapsulating Security Payload (ESP) Can provide confidentiality, authentication, integrity and replay protection

Authentication Header (AH) Authentication, integrity and replay protection, but no confidentiality and is now deprecated

Internet Key Exchange (IKE) negotiate, create, and manage session keys in so-called *security associations*

Setting up an IPsec connection

- Key exchange uses IKEv2 protocol specified in RFC 7296 (2014)
- IKEv2 uses Diffie–Hellman protocol authenticated using signatures with public keys in X.509 certificates
- Includes *cookies* to mitigate denial-of-service attacks:
  - client must return a time-dependent cookie value before the server proceeds
  - they provide *proof of reachability* before any expensive cryptographic processing is completed

Security associations

- A security association (SA) contains info needed by an IPsec endpoint to support an IPSec connection
- Can include cryptographic keys and algorithms, key lifetimes, security parameter index (SPI), and security protocol identifier (ESP or AH)
- SPI is included in the IPSec header to associate a packet with the appropriate SA
- SA tells the endpoint how to process inbound IPSec packets or how to generate outbound packets
- SAs are needed for each direction of connection
- IKEv2 is used to establish keys to use in SAs

Cryptographic suites

- Similar to TLS ciphersuites, there are a number of standardised cryptographic suites, incorporating both public key and symmetric key algorithms
- Specific groups available for Diffie–Hellman, both in finite fields and on elliptic curves
- 3DES or AES can be used for encryption, either in CBC or GCM
- HMAC or CMAC (variant) is used for integrity if GCM mode is not used

IPsec modes of operation

- Each protocol (ESP or AH) can operate in transport or tunnel mode
- Transport mode: Maintains IP header of the original packet and protects payload — generally only used in host-to-host architectures
- Tunnel mode: Original packet encapsulated into a new one, payload is original packet — typical use is gateway-to-gateway architecture
- We show the pictures for IPv4 — there are slight differences for IPv6

IPsec protocol components

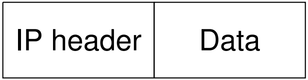
ESP header Contains the security parameter index (SPI) identifying the SA and sequence numbers

ESP trailer Contains padding and padding length — may also include extra padding to enhance traffic flow confidentiality

ESP Auth Contains MAC of the encrypted data and ESP header — may not be required if an authenticated encryption mode is used

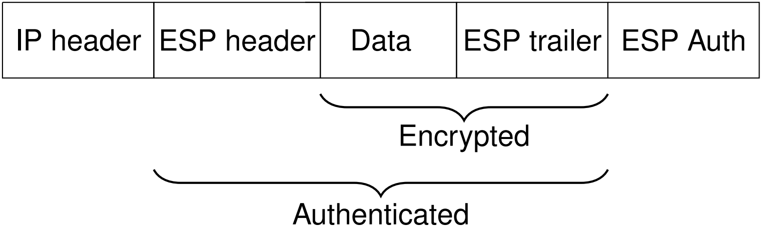
Transport mode ESP

- Original IP packet



IP header | Data

- IP Packet protected by Transport-ESP



IP header | ESP header | Data | ESP trailer | ESP Auth

Encrypted

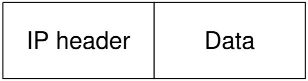
Authenticated

ESP in transport mode: Outbound packet processing

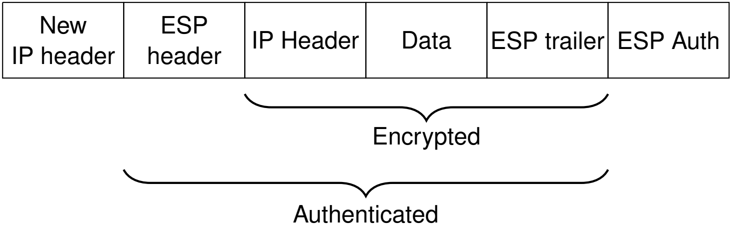
- Data after the original IP header is padded by adding an ESP trailer and result encrypted using the symmetric cipher and key in the SA
- An ESP header is prepended
- If an SA uses the authentication service, an ESP MAC is calculated over the data prepared so far and appended
- Original IP header is prepended, but some fields in the original IP header must be changed:
  - protocol field changes from TCP to ESP
  - total length field must be changed to reflect the addition of the ESP header
  - checksums must be recalculated

Tunnel mode ESP

- Original IP packet



- IP Packet protected by Tunnel-ESP



ESP in tunnel mode: Outbound packet processing

- Entire original packet is padded by adding an ESP trailer and the result encrypted using the symmetric cipher and key agreed in the SA
- ESP header is prepended
- If the SA uses the authentication service, an ESP MAC is calculated over the data prepared so far and appended
- New outer IP header is prepended
  - Inner IP header of the original IP packet carries the ultimate source and destination addresses
  - Outer IP header may contain distinct IP addresses such as addresses of security gateways
  - Outer IP header protocol field is set to ESP

IPsec security

- Active attacks have been demonstrated for *encryption-only* mode of ESP protocol — now widely understood that providing encryption without integrity is insecure
- Unlike earlier versions of IPsec, the 2005 version does not require implementations to support encryption-only mode, but still allows it
- ESP applies encryption before MAC in normal usage
- Using AH, a MAC can be applied before encryption, as in TLS. Attacks have been demonstrated on such configurations
- Formal analysis has shown that IPsec key exchange protocol (IKEv2) has no significant weaknesses

Lecture 15: Email Security and Secure Messaging

TTM4135

Relates to Stallings Chapter 19.
Stallings does not cover secure messaging

Spring Semester, 2025

Motivation

- Email remains one of the most widely used forms of electronic communication but is often sent without end-to-end security
- Instant messaging is increasingly popular and has been built with good security
- Both use cryptography extensively but in practice have very different security propeties

Outline

- Email Security
  - Email Security Requirements
  - Link Security
  - End-to-end Security
    - PGP
    - S/MIME
- Secure Messaging

Email architecture

- Message user agent (MUA) connects client to mail system. Uses SMTP to send mail to message submission agent (MSA) and POP or IMAP to retrieve mail from message store (MS).
- Message handling system (MHS) transfers message from MSA to MS via one or more message transfer agent (MTA)
- Simple message transfer protocol (SMTP) is mail transmission protocol defined in RFC 5321
- Today it is very common to use *webmail* which is a browser interface to an online email client. Note that SMTP and POP/IMAP are still used to send and receive email

Email architecture in a picture

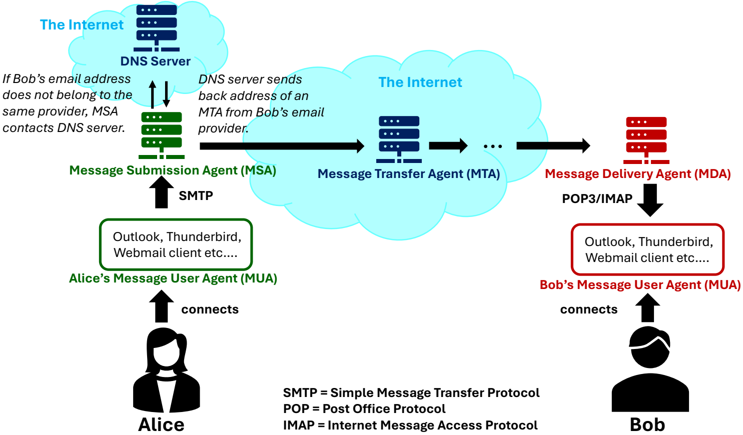

The Internet
DNS Server
If Bob's email address does not belong to the same provider, MSA contacts DNS server.
DNS server sends back address of an MTA from Bob's email provider.
The Internet
Message Submission Agent (MSA)
Message Transfer Agent (MTA)
...
Message Delivery Agent (MDA)
SMTP
POP3/IMAP
Outlook, Thunderbird, Webmail client etc....
Alice's Message User Agent (MUA)
Outlook, Thunderbird, Webmail client etc....
Bob's Message User Agent (MUA)
connects
connects
Alice
Bob
SMTP = Simple Message Transfer Protocol
POP = Post Office Protocol
IMAP = Internet Message Access Protocol

Figure: Email architecture

Security threats against email

- We may consider threats in the usual ‘CIA’ categories
- Email content may require confidentiality or authentication
- Availability of the email service may be threatened
- Metadata in header information is a significant source of attacker information

Spam

- Unsolicited (bulk) email
- A cheap form of advertising?
- Common vector for phishing attacks
- Countermeasures typically use email filtering
- Phishing with more accurate targeting (spear phishing) is harder to filter

Link security and end-to-end security

- Security may be provided between different agents in the mail system on a link-by-link basis using protocols such as STARTTLS and DKIM
- Alternatively it may be provided from client to client (end-to-end) using protocols such as PGP and S/MIME
- Both have their advantages and disadvantages. Ideally both are used.

Link security and end-to-end security in a picture

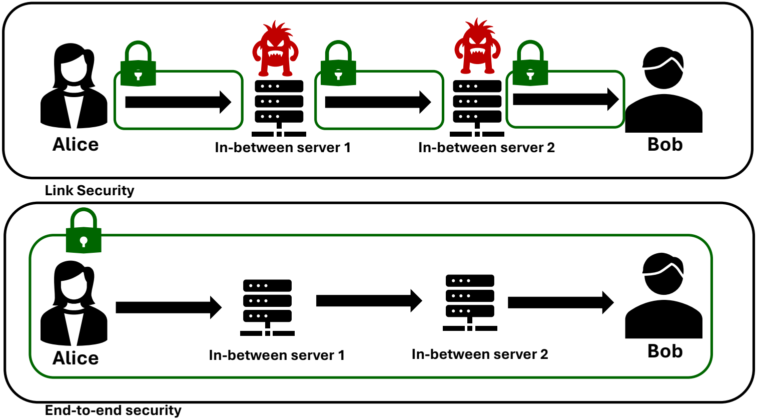


Figure: Link Security vs End-to-end Security

STARTTLS

- Extensions to mail protocols SMTP, POP and IMAP to run over TLS connections
- Provides link-by-link security, not end-to-end security
- *Opportunistic* use of TLS security (encryption) — use it if possible
- Defined for IMAP and POP3 (RFC 2595) and for SMTP (RFC 3207) amongst other protocols
- Widely used by prominent email providers including Gmail and Microsoft Outlook
- Vulnerable to so-called STRIPTLS attacks – attacker interrupts TLS negotiation and connection falls back to plaintext transmission.

DomainKeys Identified Mail (DKIM)

- Standardised in RFC 6376 (2011)
- Allows sending mail *domain* to sign outgoing mail using RSA signatures (currently supported signature algorithm)
- Receiving *domain* can verify origin of mail
- Widely used by prominent email providers including Gmail
- Helps prevent email spoofing and hence reduce spam and phishing
- Example on next slide shows 2048 bit RSA signature on message, coded in base64
- Public verification key of sending domain retrieved using DNS

Example DKIM signature

```
v=1;   // Version
a=rsa-sha256;  // Algorithm
c=relaxed/relaxed; // Header/body canonicalization (format)
d=easychair.org;  // Domain claiming origin
s=default;        // Selector subdividing namespace
t=1677503401; // Timestamp
h=Content-Type:Date:From:Subject:Sender:From;
// Signed header fields
bh=L56upQ4J/BTdlVqCi3PP+Ab67CIehSnUzUFm1aRFEIg=;
// Hash of the body part
b=cS0GpBApvz1YTNs93xkduJgryOnEp/l/t+TAvRFb0HL16ACrttSdnN
UoMVT1se1ZxPpqff9DaAW5DSeBrm5CQUfJvnf8Q7e2ZvJGukpJiiRn
NfNCVy5TIxI5N1oDXCeUT8q kn/YcyxzOjpF+8mmzFo4aK/5NQD/jT1
/Ydfwl/jegHB0c9+rNPHgtlJd7ANOc+GNgS XCHIYL4jhMTnCN4VNM
sgBLQMhFcfU0rWbNaX6Z37r9PwvEli+MpXzYHL68do9skO8B O60Y
Z9MOG8vI1ara40DIuRTVdK3d45geYOTy3rp55VbKC/kY4AKMCCwm
dFgMl75KY7 f5QCpWUhpogEQ==      // Signature
```

DKIM public keys

- The ’d=’ and ’s=’ parts of the DKIM signature specify domain and selector
- The relevant public key is in the DNS record for the host defined by the host name: `[selector]._domainkey.[domain]` where
  - ’s=’ value is the selector
  - ’d=’ value is the domain
- In the example header above the nslookup would be:

```
nslookup -type=txt default._domainkey.easychair.org
```

Take-up of DKIM and STARTTLS

- In February 2023, Gmail was using STARTTLS for around 90% of both outgoing and incoming emails. (`https://www.google.com/transparencyreport/saferemail/?hl=en`)
- A 2020 study found that just under 60% of emails included a DKIM signature: Georgios Kambourakis and others. *What Email Servers Can Tell to Johnny: An Empirical Study of . . . Email Security*, IEEE Access, July 2020.
- The same study noted around 97% usage of STARTTLS

History of PGP

- Originally product of one person — Phil Zimmermann
- Subject of widely reported export restriction controversy
- OpenPGP standard, specified in RFC 4880, allows for interoperable implementations
- GnuPG (GPG) is an open implementation.
- PGP corporation acquired by Symantec in 2010

Photo: User Matt Crypto on en.wikipedia

Email processing

- Protection of email message contents
- Hybrid encryption — a new random “session key” is generated for each object (message) and the session key is encrypted with the long-term public key of recipient
- Signing using RSA or DSA signatures
- Compression using Zip
- Coding using base-64 to ensure that binary strings can be sent in email body

PGP encryption

- Session keys are encrypted using asymmetric encryption. OpenPGP requires support for ElGamal encryption and recommends also to support RSA encryption.
- Encryption of message text using symmetric key encryption – OpenPGP requires support for 3DES with three keys (168 bits in total) and recommends also AES-128 and CAST5. Other algorithms are also defined.
- Compression is applied before encryption
- Encryption can be applied independently of signing (no requirement for authenticated encryption)

PGP signatures

- Plaintext message is optionally signed with sender’s private key
- OpenPGP standard requires support for RSA signatures
- DSA signatures also defined
- RSA signed messages are hashed with SHA1 (support required in standard) or other SHA2 hash functions

OpenPGP PKI

- Used in PGP email security
- Includes ID, public key, validity period and a *self-signature*
- No certification authorities — keys can be signed by anyone
- Various key servers used to store keys, such as `https://keys.openpgp.org/`
- Often known as the *web of trust*

Web of Trust in Pictures

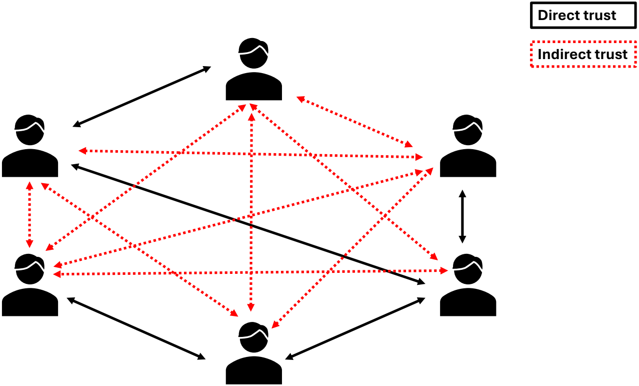


Figure: Schematic Representation of the Web of Trust

Central Authority in Pictures

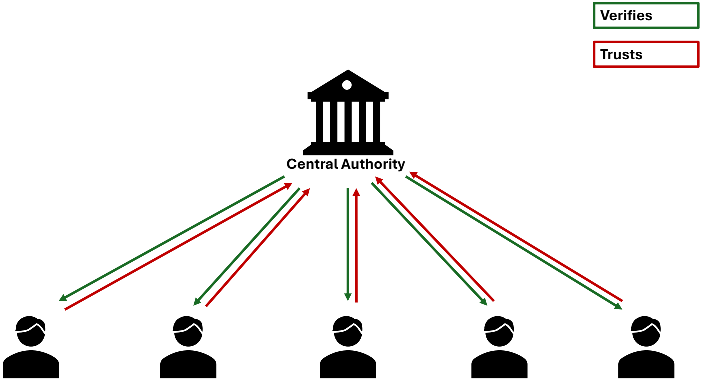


Figure: Schematic Representation of a Central Authority

Usability

- Can we expect the average user to understand public key cryptography?
- Is it possible to design a PGP interface that helps users to operate PGP correctly and safely?
- See: Alma Witten and J. D. Tygar, *Why Johnny can't encrypt: A Usability Evaluation of PGP 5.0*, 1999
- Follow-up studies show that newer PGP versions are still hard to use
- Typical problems:
  - Generating new keys securely
  - Moving keys between devices
  - Renewing keys when they expire

Take-up of PGP

- Plugins available for many popular mail clients and for webmail interfaces (Mailvelope, OpenKeyChain) (see list at `https://www.openpgp.org/software/`)
- Some mailer servers, such as ProtonMail, provide compatibility but manage your private key for you
- The key server `https://keys.openpgp.org/about` was launched in June 2019 and has currently around 350000 keys

Criticisms of OpenPGP

- Outdated cryptographic algorithms still used: SHA1, CAST, Blowfish, . . .
- No support for SHA3 or authenticated encryption such as GCM
- A lot of metadata is available to an eavesdropper including
  - file length
  - encryption algorithm used
  - key identity of recipients
- No forward secrecy
- Does not support streaming mode or random access decryption

S/MIME

- Similar security features to PGP but different format for messages and not interoperable
- Requires X.509 format certificates instead of web of trust
- Supported natively by most popular mail clients

Differences between email and messaging

Email and messaging have obvious similarities but also important differences

- Most instant messages are part of an interactive conversation which extends over many messages and a long time
- Proprietary servers are typically used to manage accounts and dedicated applications are used

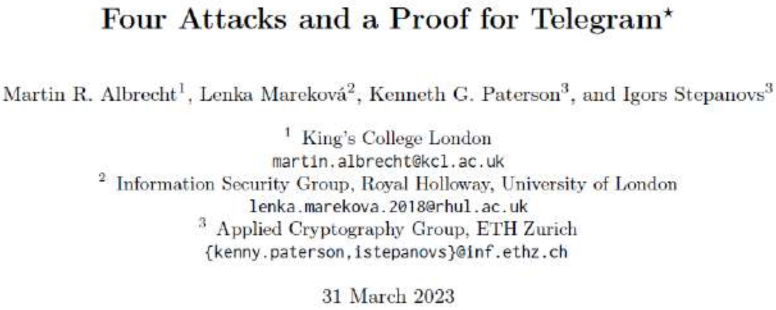
Messaging security

- The standard CIA security services are important as usual
- Forward secrecy is important especially for long sessions — achieved using *medium-term* public keys stored at the server
- Desirable also to have *post-compromise security* (self-healing): an attacker who obtains a long-term key should be locked out again after communication resumes

Messaging security standards

- There is no standardized (secure) messaging protocol
- Different apps do security in different ways - with varied levels of success (see Wikipedia comparison)
- Snapchat, Discord: no End-to-End encryption
- (Facebook) Messenger: End-to-end encryption since April 2024.
- iMessage, and Whatsappare (allegedly) secure.
- Telegram only offers encrypted chat, if a secret chat is opened, the normal chats are not by default encrypted. Additionally, they use a custom encryption protocol.
- Signal is generally considered the most secure and is open source

Attacks on Telegram



Four Attacks and a Proof for Telegram*

Martin R. Albrecht[1], Lenka Mareková[2], Kenneth G. Paterson[3], and Igors Stepanovs[3]

[1] King's College London
martin.albrecht@kcl.ac.uk
[2] Information Security Group, Royal Holloway, University of London
lenka.marekova.2018@rhul.ac.uk
[3] Applied Cryptography Group, ETH Zurich
{kenny.paterson,istepanovs}@inf.ethz.ch

31 March 2023

Figure: Paper from 2023 that shows major attacks on Telegram

Signal protocol

- Signal server sets up initial authentication of user and registers initial public keys
- Public keys at the server are used to set up initial communication between users
- Key exchange uses elliptic curve Diffie–Hellman
- AES in CBC mode with HMAC (SHA256) used for message protection
- Protocol is used in Signal app and claimed also to be in WhatsApp and Facebook Messenger (closed source)

Ratcheting

- A ratchet is a device which is easy to move forward but blocked from moving backward
- Signal uses a new unique message key for every message exchanged, known as *continuous key exchange*
- When successive messages sent in the same direction the message key is updated with a *symmetric ratchet* by applying a function such as HMAC
- When a new message is returned in the opposite direction a new Diffie-Hellman ephemeral key is used to compute the new message key: this is the *Diffie-Hellman ratchet*
- Many more details in the online specification: `https://signal.org/docs/specifications/doubleratchet/`

Group messaging

- No good alternative for Diffie-Hellman is known in the mutli-party case
- Signal uses a simple key distribution method for group messaging
- Currently a research effort is under way to develop Messaging Layer Security (mls) standard: https://datatracker.ietf.org/wg/mls/about/