COMPSCI 4CR3 - Applied Cryptography

Jake Doliskani

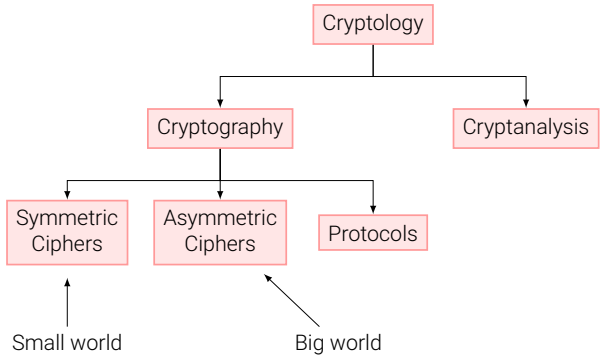
McMaster University

Introduction to Cryptography

This lecture

- Overview on the field of cryptology
- Basics of symmetric cryptography
- Cryptanalysis
- Substitution Cipher
- Shift (or Caesar) Cipher and Affine Cipher

Overview


Cryptology
Cryptography
Cryptanalysis
Symmetric Ciphers
Asymmetric Ciphers
Protocols
Small world
Big world

Cryptography

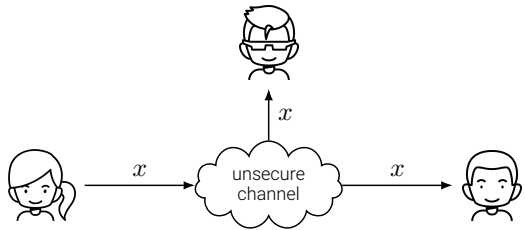
is the science/art of protecting secrets.

Pre-modern cryptography

- Very old
  Scytale of Sparta, Caesar cipher

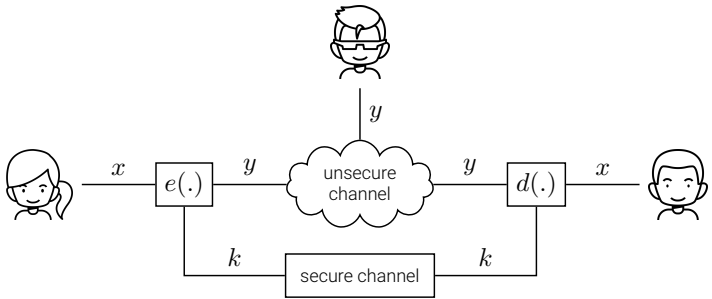
- More recent
  Enigma

Symmetric Cryptography



- Alice and Bob want to communicate over an unsecure channel
- Trudy has access to the channel but should not be able to understand the messages

Symmetric Cryptography



- x is the plaintext
- y is the ciphertext
- k is the key

Symmetric Cryptography

- The encryption $y = e_k(x)$ and decryption $x = d_k(y)$ must be inverse to each other:

$$d_k(e_k(x)) = x \text{ for all messages } x.$$

- The encryption and decryption algorithms are public; everyone knows them.
- The system is only secure if the key is shared between Alice and Bob in a secure way
  - This can be done by public key cryptography (later in the course)

The problem of secure communication is reduced to secure transmission and storage of the key.

The Substitution Cipher

- Substitute symbols using a permutation.
- The permutation is the key
- Alphabet example:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	,	‘	;	.	?
y	e	q	j	u	l	’	w	s	r	p	i	f	x	t	h	m	.	z	k	,	d	;	v	o	a	b	g	?	c	n

`information` ⟹ `sxt.fykstx`

The Substitution Cipher

Attacks

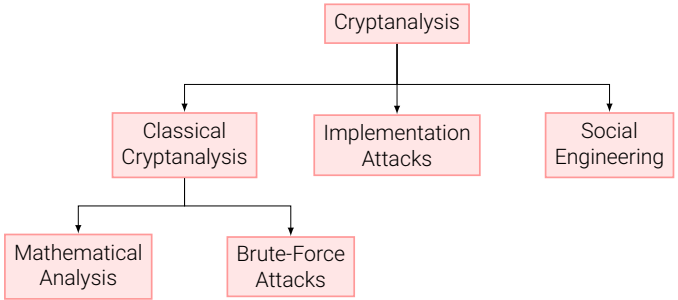
- Brute-Force
  - Try all possible keys
  - Slow; for example, for 31 symbols, there are $\approx 2^{112}$ keys.
- Letter Frequency Analysis
  - Determine the frequency of every ciphertext letter
  - Compare with the frequency distribution of the language

```
iq ifcc vqqr fb rdq vfllcq na rdq cfjwhwz hr bnnb
hcc hwwhbsqvqbre hwq vhlq
```

```
We will meet in the middle of the library at noon
all arrangements are made
```

e	0.12702	m	0.02406
t	0.09056	w	0.0236
a	0.08167	f	0.02228
o	0.07507	g	0.02015
i	0.06966	y	0.01974
n	0.06749	p	0.01929
s	0.06327	b	0.01492
h	0.06094	v	0.00978
r	0.05987	k	0.00772
d	0.04253	j	0.00153
l	0.04025	x	0.0015
c	0.02782	q	0.00095
u	0.02758	z	0.00074

Cryptanalysis



- **Implementation attacks:** try to extract the key using side channel techniques, e.g. power analysis.
- **Social engineering:** try to obtain the information about the key using social interactions.

Why do we need cryptanalysis?

- There is no (and probably will never be) mathematical proof of security for any practical cipher
- The only way to have assurance that a cipher is secure is to try to break it (and fail)!

Only use cryptosystems that are well established, i.e., have been cryptanayzed for many years.

Kerckhoffs' Principle

A cryptosystem should be secure even if the attacker knows all details about the system, with the exception of the secret key. In particular, the system should be secure when the attacker knows the encryption and decryption algorithms.

How Many Key Bits Are Enough?

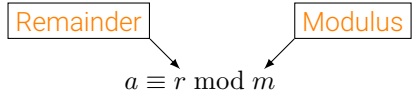
- Key length is estimated based on the best known attack
- Key lengths for symmetric and asymmetric algorithms are dramatically different

Example: for successful brute-force attacks on symmetric algorithms:

Key length	Security estimation
56–64 bits	short term: a few hours or days
112–128 bits	long term: several decades in the absence of quantum computers
256 bits	long term: several decades, maybe even with quantum computers

Modular Arithmetic

Modulo operation:



- It means $m \mid a - r$ (m divides $a - r$)
- Equivalently, $a = mq + r$ for some integer q.

Example: $73 \equiv 8 \bmod 13$.

- The remainder r is not unique: $73 \equiv 21, -18 \bmod 13$
- There is an infinite set of such remainders: $\{\ldots, -18, -5, 8, 21, \ldots\}$
- This is called an equivalence class. We can use any of the members in this equivalence class in operations modulo 13.

We usually choose r such that $0 \leq r < m$.

Modular Arithmetic

- We compute modular division by multiplying by the inverse, i.e. $a/b \bmod m$ is $ab^{-1} \bmod m$.
  - The inverse b^{-1} of b satisfies $bb^{-1} \equiv 1 \bmod m$.
  - Example: to compute $3/5 \bmod 9$, we first find $5^{-1} \equiv 2 \bmod 9$ and then compute $3 \cdot 2 \equiv 6 \bmod 9$.

- The inverse b^{-1} exists only if $\gcd(b, m) = 1$.
  - In this case, we say b and m are coprime.
  - We sometimes write $(b, m) = 1$ instead

Modular Arithmetic

- Modular reduction can be performed at any point of computation
- Example: exponentiation
  - Modular reduction at the end:

$$7^5 = 16807 \equiv 11 \bmod 13$$

  - Modular reduction throughout:

$$7^5 = 7^2 \cdot 7^2 \cdot 7 \equiv 10 \cdot 10 \cdot 7 \equiv 11 \bmod 13$$

It is usually computationally cheaper to perform intermediate modular reduction.

Modular Arithmetic (Integer Rings)

The integer ring is the set $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$ on which we define two operations:

- Multiplication: $a \times b$ is defined as $a \times b \bmod m$ for all $a, b \in \mathbb{Z}_m$.
- Addition: $a + b$ is defined as $a + b \bmod m$ for all $a, b \in \mathbb{Z}_m$.

Example: $\mathbb{Z}_7 = \{0, 1, \ldots, 6\}$ with addition and multiplication $\bmod 7$.

Integer rings are special cases of general rings.

Modular Arithmetic (Integer Rings)

A ring R is a set with the following properties for all $a, b, c \in R$:

- Closure: The results of addition and multiplication are always in R.
- Associativity:

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

- Distributive law: $a \times (b + c) = (a \times b) + (a \times c)$
- Neutral element for addition: $a + 0 = a$
- Neutral element for multiplication: $a \times 1 = a$
- Additive inverse: $a + (-a) = 0$
- Multiplicative inverse: $a \times a^{-1} = 1$ (might not always exists)

The Caesar Cipher

Shift each symbol (and wrap around)

- Key: $k \in \mathbb{Z}_m$
- Plaintext: $x \in \mathbb{Z}_m$
- Ciphertext: $y \in \mathbb{Z}_m$

Encryption: $e_k(x) \equiv x + k \bmod m$
Decryption: $d_k(y) \equiv y - k \bmod m$

Example: the alphabet

- $m = 26$
- Key: $k = 17$

Palintext: `attack` $= 0, 19, 19, 0, 2, 10$.
Ciphertext: $17, 10, 10, 17, 19, 1 =$ `rkkrtb`

The Affine Cipher

Transform each symbol by a linear map

- Key: $a, b \in \mathbb{Z}_m$, where a is invertible $\bmod m$
- Plaintext: $x \in \mathbb{Z}_m$
- Ciphertext: $y \in \mathbb{Z}_m$

Encryption: $e_k(x) \equiv ax + b \bmod m$
Decryption: $d_k(y) \equiv a^{-1}(y - b) \bmod m$

Example: the alphabet

- $m = 26$
- Key: $(a, b) = (9, 13)$, where $a^{-1} \equiv 3 \bmod 26$

Palintext: $\texttt{attack} = 0, 19, 19, 0, 2, 10$.
Ciphertext: $13, 2, 2, 13, 5, 25 = \texttt{nccnfz}$

COMPSCI 4CR3 - Applied Cryptography

Jake Doliskani

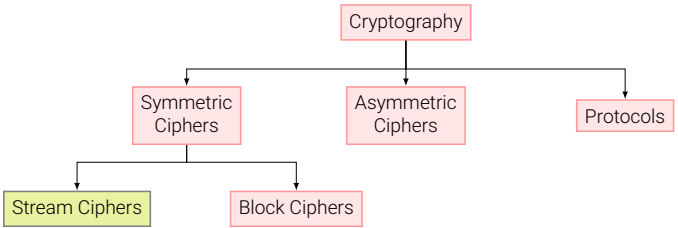
McMaster University

Stream Ciphers

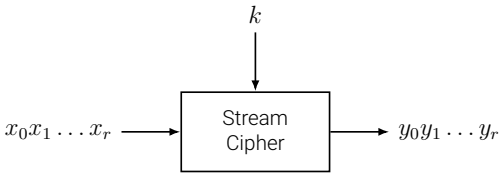
This lecture

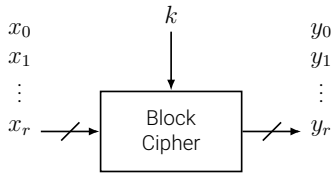
- Introduction to stream ciphers
- Random and pseudorandom number generators
- The One-Time Pad (OTP)
- Linear feedback shift registers

Stream Ciphers



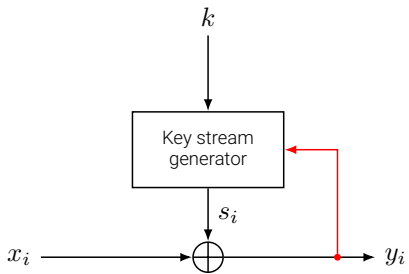
Stream vs Block Ciphers





- Stream cipher: 1 bit at a time
  - Xor 1 bit of the key stream with 1 bit of the input
- Block cipher: 1 block at a time
  - Encode an entire block using the same key

Stream cipher: synchronous vs asynchronous



- **Synchronous:** key stream depends on the key
- **Asynchronous:** key stream depends on the key and the ciphertext
- Most practical stream ciphers are synchronous

Block ciphers

- Encrypt an entire block of plaintext bits at a time
- Example block lengths:
  - 128 bits in AES
  - 64 bits in DES

- In practice, block ciphers are used more often than stream ciphers.
- Stream ciphers are small and fast, they are used smaller devices with less resources.
- Traditionally, it was assumed that stream ciphers are more efficient. Modern block ciphers are now very fast.

Stream Ciphers: Encryption, Decryption

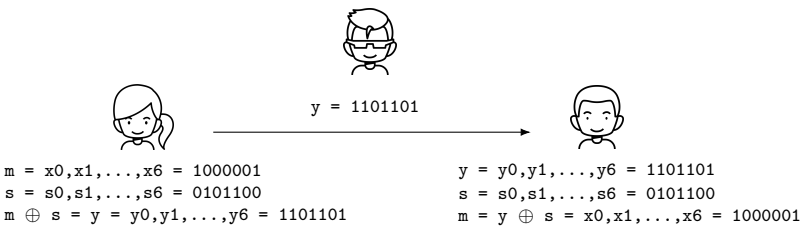
$x_i, y_i, s_i \in \{0, 1\}$
Encryption: $y_i = e(s_i, x_i) = x_i + s_i \bmod 2$
Decryption: $x_i = d(s_i, y_i) = y_i + s_i \bmod 2$

Encryption and Decryption are the same functions

Why addition mod 2?

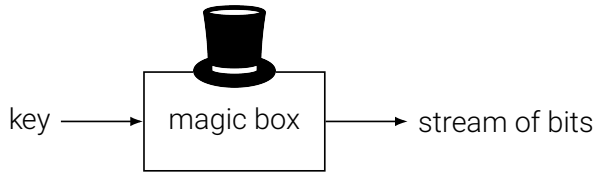
- The same as XOR
- If the key is random, it is impossible for Trudy to extract the plaintext from the ciphertext.

Example:



The key stream

- The security of the stream cipher depends completely on the key stream.
- It is an algorithm that generate an unbounded number of secret bits.
- The actual key is used as an input/seed
- The more “random looking” the bits the better.



Random Number Generators

- True Random Number Generators
  - The output cannot be reproduced
- How can we build them?
- Maybe using a quantum computer?

- Pseudorandom Number Generators (PRNG)
  - Uses a function for generation
  - Uses a seed
- Example:

  $s_0 =$ seed

  $s_{i+1} \equiv as_i + b \pmod{m}$

  The `rand()` function in ANSI C:

  $s_0 = 12345$

  $s_{i+1} \equiv 1103515245 s_i + 12345 \pmod{2^{31}}$

Cryptographically secure PRNG

- A PRNG which is unpredictable.
  - Given a bunch of bits of the stream, it is computationally infeasible to compute the subsequent bits.
  - More precisely: for some parameter n, given a polynomials (in n) number of consecutive bits of the stream, there is no polynomial time (in n) algorithm that can predict the next bit.

Do cryptographically secure PRNGs exist?

The One-Time Pad

Unconditional Security

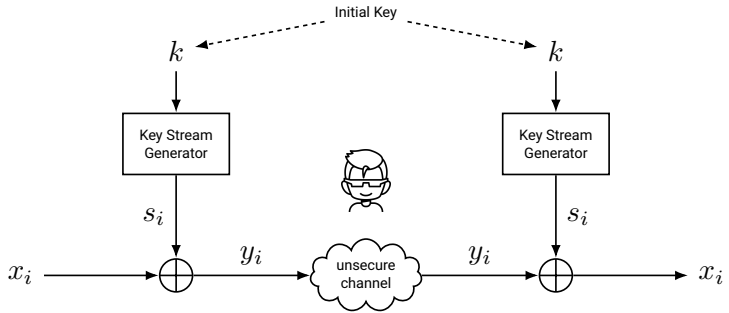
A cryptosystem is unconditionally (or information-theoretically) secure if it cannot be broken even with infinite computational resources.

One-Time Pad (OTP)

1. The key stream $s_0, s_1, \ldots$ is generated by a true random number generator
2. Every key stream bit s_i is used only once

OTP is Unconditionally Secure.

Practical stream ciphers



Computational Security

A cryptosystem is computationally secure if there is no known polynomial time algorithm that can break it.

Key streams from PRNGs

- Many PRNGs have good statistical properties, e.g., the output bit stream look random
- Example: for the key $k = (a, b)$,

  $s_0 = \text{seed}$
  $s_{i+1} \equiv as_i + b \bmod m$

- But, is looking random good enough?

- Attack: If Trudy obtains

  $s_2 \equiv as_1 + b \bmod m$
  $s_3 \equiv as_2 + b \bmod m$

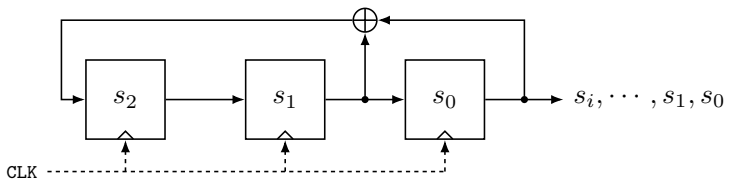
- Then he can easily solve for $a, b \bmod m$.

We need cryptographically secure PRNGs.

Shift Register-Based Stream Ciphers

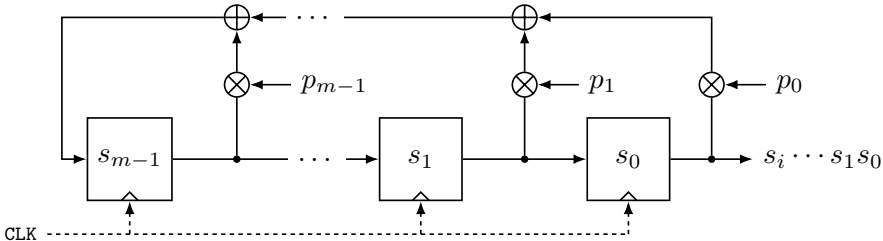
- Linear Feedback Shift Registers (LFSR)
  - An LFSR consists of clocked storage elements (flip-flops) and a feedback path
  - The number of storage elements is the degree of the LFSR

Example:



$$s_3 = s_1 + s_0 \bmod 2$$

General LFSRs



- If $p_i = 1$, the feedback is active. If $p_i = 0$, the feedback is inactive.
- Mathematically:

$$s_m = p_{m-1}s_{m-1} + \cdots + p_1 s_1 + p_0 s_0 \bmod 2$$

LFSRs are periodic

- Mathematically, LFSRs are described by linear recurrences (as we just saw)
- After a finite number of output bits the sequence will repeat
- The length of the period is called the length of the LFSR

Example: the LFSR $s_3 = s_1 + s_0 \bmod 2$ has length 7. For the initial value $(s_0 = 0, s_1 = 0, s_2 = 1)$, the output is `0010111 0010111 0010111 ...`

Theorem

The maximum length of an LFSR of degree m is $2^m - 1$.

LFSRs for stream ciphers?

- If we use an LFSR as a stream cipher, the secret key is the set of coefficients $(p_{m-1}, \ldots, p_1, p_0)$.
- If Trudy knows some plaintexts, he can lunch an attack called the **Known-Plaintext Attact.**
- Suppose Trudy knows the plaintext bits

$$x_0, x_1, \ldots, x_{2m-1}$$

and the corresponding ciphertext bits

$$y_0, y_1, \ldots, y_{2m-1}$$

Then he can compute the first $2m$ key stream bits:

$$s_i = x_i + y_i \bmod 2, \qquad i = 0, 1, \ldots, 2m-1.$$

Known-Plaintext Attack

The recurrence relation for the LFSR is

$$s_{i+m} = p_{m-1}s_{i+m-1} + \cdots + p_1 s_{i+1} + p_0 s_i \bmod 2, \quad s_i, p_j \in \{0,1\}, \quad i = 0, 1, \ldots$$

Having $2m$ key bits, Trudy obtains

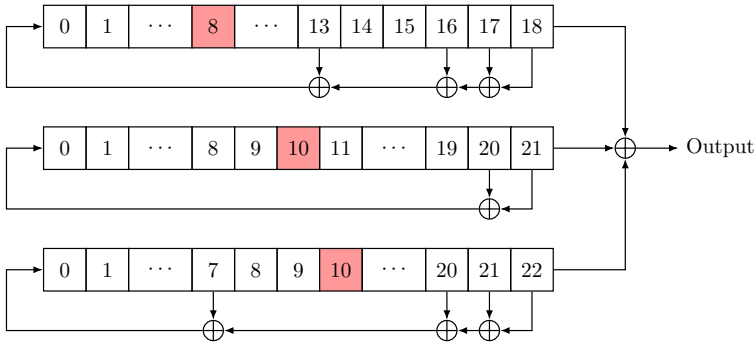
$$\begin{array}{lll} i=0 & s_m & = p_{m-1}s_{m-1} + \cdots + p_1 s_1 + p_0 s_0 \mod 2 \\ i=1 & s_{m+1} & = p_{m-1}s_m + \cdots + p_1 s_2 + p_0 s_1 \mod 2 \\ \vdots & \vdots & \vdots \\ i=m-1 & s_{2m-1} & = p_{m-1}s_{2m-2} + \cdots + p_1 s_m + p_0 s_{m-1} \mod 2 \end{array}$$

- A system of linear equations $\bmod 2$!
- So, what should we do?
  $\Rightarrow$ Use multiple LFSRs with nonlinear components

Multiple LFSRs

- Use nonlinear components to avoid attacks like the one we just saw

  Example: A5/1



COMPSCI 4CR3 - Applied Cryptography

Jake Doliskani

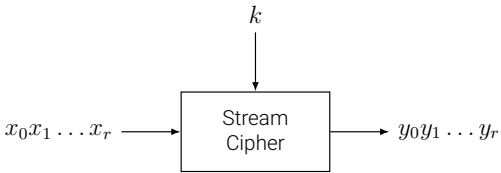
McMaster University

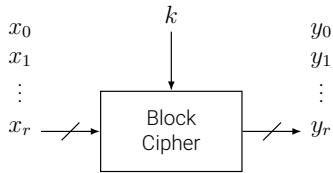
Block Ciphers

This lecture

- Introduction to block ciphers
- A brief review of DES
- Overview of the AES
- The internal structure of AES
- Practical issues

Stream vs Block Ciphers





- Stream cipher: 1 bit at a time
  - Xor 1 bit of the key stream with 1 bit of the input
- Block cipher: 1 block at a time
  - Encode an entire block using the same key

Block Ciphers

Desired properties:

- Detecting any relations between the key and ciphertexts should be infeasible.
- Ciphertext should not reveal any statistical properties of the plaintext.
  - Any change, even a single bit, in the plaintext should have a “random” effect on all the bits of the ciphertext.

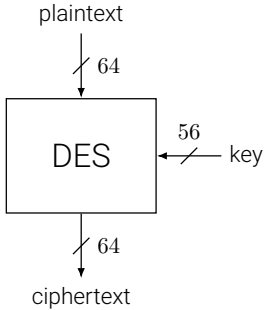
Idea: Product ciphers.

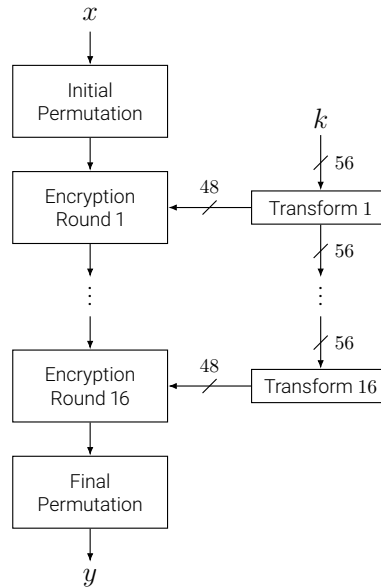
- Apply many rounds of some carefully designed function to the input.

The Data Encryption Standard (DES)

- DES encrypts blocks of size 64 bits and has key length of 56 bits.
- The National Institute of Standards and Technology (NIST) initiated a call for proposal for a standardaized encryption scheme in 1972
- A promissing candidate was submitted by IBM in 1974, which was based on a scheme called Lucifer.
  - Lucifer was a family of ciphers developed by Horst Feistel in the late 1960s.
- The final design of DES was influenced by National Security Agency (NSA)
  - Cryptographers suspected that NSA could have added a secrete Backdoor to DES.
- In 1977, NIST released the complete specifications of DES, named as the Data Encryption Standard.

DES overview

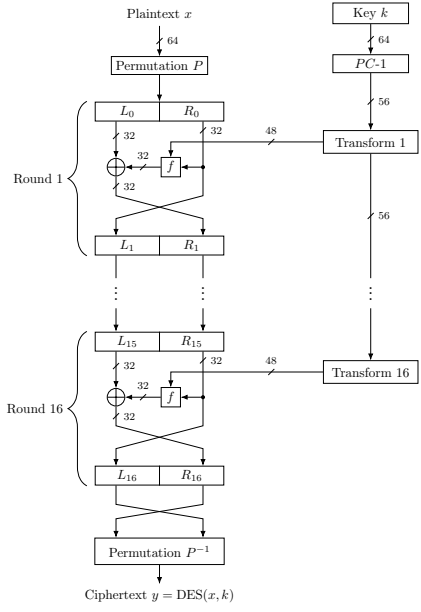




The DES Feistel Network

- The DES structure is a Feistel network
- In a Feistel network, encryption and decryption are mostly the same procedures.
- Each round is given by:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$$



Security of DES

- **Analytical Attacks:** No known major analytical attacks (That is true most block ciphers)

- Two major criticisms:
  1. Key space is too small (2^{56} possible keys). An exhaustive key search on a modarn machine can easily break DES.

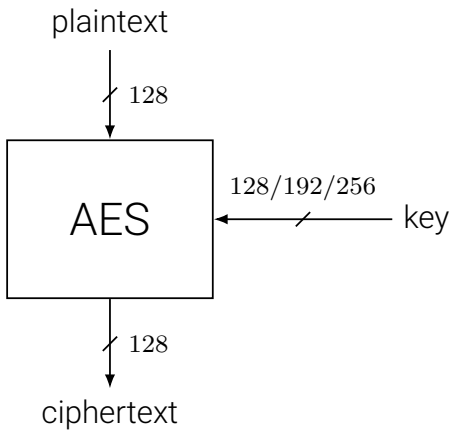
     Given a pair (x, y) of plaintext and ciphertext, test all $k \in \{0,1\}^{56}$ until $\mathrm{DES}(x, k) = y$.

  2. The S-box design criteria have been kept secret (to this day!). There might be backdoors only known to the NSA.

Enters AES

- The The Advanced Encryption Standard (AES) is the most widely used symmetric cipher today
- NIST initiated a call for proposals for a new block cipher in January, 1997
- Out of 15 candidates 5 finalists were announced in August, 1999
  - Mars - IBM Corporation
  - RC6 - RSA Laboratories
  - Rijndael - J. Daemen & V. Rijmen
  - Serpent – Biham et al.
  - Twofish - Schneier et al.
- In October 2000, Rijndael was chosen as the AES standard

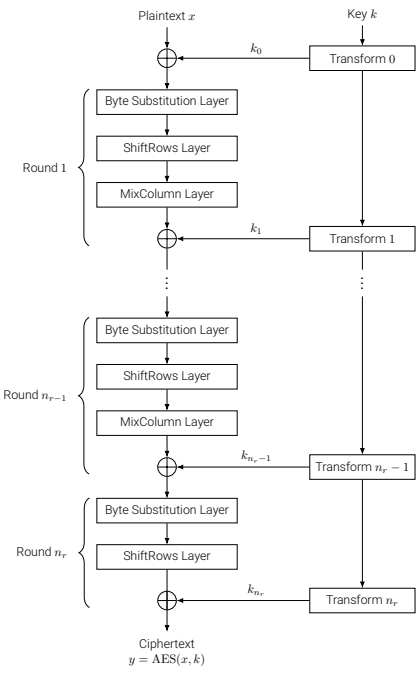
Overview of AES



Key length (bits)	Rounds $= n_r$
128	10
192	12
256	14

Overview of AES

- AES is not a Feistel network
- Number of rounds $n_r = 10$ or 12 or 14 based on key size
- Each round consists of layers



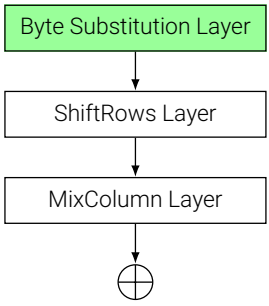
Internal Structure of AES

- AES is a byte-oriented cipher
- The input of AES can be arranged in a 4×4 matrix

$$A = \begin{bmatrix} A_0 & A_4 & A_8 & A_{12} \\ A_1 & A_5 & A_9 & A_{13} \\ A_2 & A_6 & A_{10} & A_{14} \\ A_3 & A_7 & A_{11} & A_{15} \end{bmatrix}$$

  where each A_j is a byte, and $A_0, A_1, \ldots, A_{15}$ is the 16-byte input of AES.
- AES operates on a 4×4 matrix like this, called the data state, throughout its exectution.

Internal Structure of AES

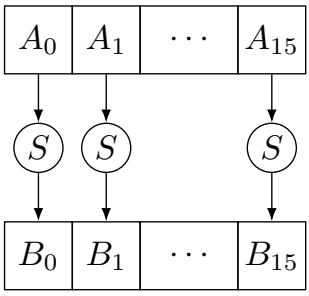


Byte Substitution Layer

- A row of 16 parallel substitution boxes (S-box).
- An S-box is a function $S : \{0,1\}^8 \to \{0,1\}^8$; It is applied to each byte: $S(A_i) = B_i$.
- The S-box S is non-linear: for two states A and A'

$$S(A_i) + S(A'_i) \neq S(A_i + A'_i).$$

- S-boxes are usually implemented using lookup tables.
- DES also uses S-boxes, but unlike AES, it uses 16 different S-boxes, one for each byte.

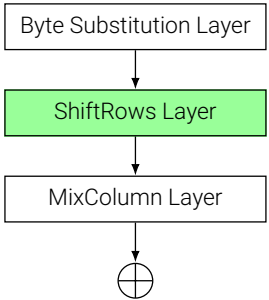


Byte Substitution Layer

AES S-box, input byte = xy

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Internal Structure of AES

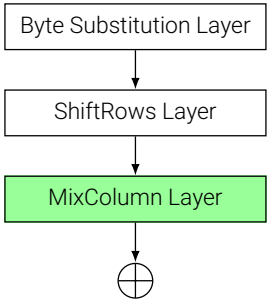


ShiftRows Layer

$$\begin{bmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_1 & B_5 & B_9 & B_{13} \\ B_2 & B_6 & B_{10} & B_{14} \\ B_3 & B_7 & B_{11} & B_{15} \end{bmatrix} \Longrightarrow \begin{bmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_5 & B_9 & B_{13} & B_1 \\ B_{10} & B_{14} & B_2 & B_6 \\ B_{15} & B_3 & B_7 & B_{11} \end{bmatrix}$$

- Row 1: no shift
- Row 2: one positions left shift
- Row 3: two positions left shift
- Row 4: three positions left shift

Internal Structure of AES



MixColumn Layer

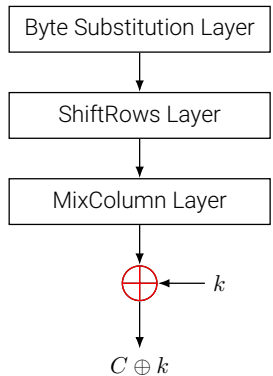
$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \overbrace{\begin{bmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_5 & B_9 & B_{13} & B_1 \\ B_{10} & B_{14} & B_2 & B_6 \\ B_{15} & B_3 & B_7 & B_{11} \end{bmatrix}}^{\text{Input}} = \overbrace{\begin{bmatrix} C_0 & C_4 & C_8 & C_{12} \\ C_1 & C_5 & C_9 & C_{13} \\ C_2 & C_6 & C_{10} & C_{14} \\ C_3 & C_7 & C_{11} & C_{15} \end{bmatrix}}^{\text{Output}}$$

- Linear transformation which mixes the columns of the state matrix
- All arithmetic is done in the Galois field $\mathrm{Gal}(2^8)$.

Internal Structure of AES

Key Addition Layer:

- Inputs:
  - 16-byte state matrix C
  - 16-byte subkey k
- Output: $C \oplus k$
- The subkeys are generated in the key schedule



Key Schedule

- A subkey is used in each round, plus one subkey at the beginning.

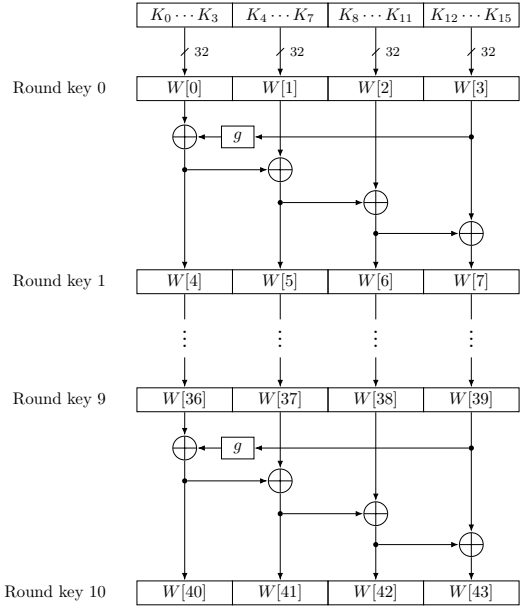
Key length (bits)	Number of subkeys
128	11
192	13
256	15

- Subkeys are derived from the original key
- The key schedule circuits are different for AES-128, AES-192 and AES-256

Key Schedule

Key schedule for AES-128:

- No byte operations, only `int` (= 4 bytes) operations
- Each iteration produces a subkey; 11 subkeys in total
- The first subkey is the original key.

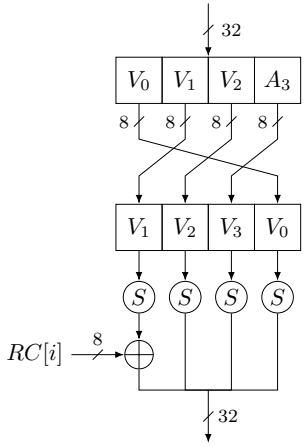


Key Schedule

The function g:

- The $RC[i]$ are called the round coefficients. They are different for each round.

```
RC[1]  = 0x01
RC[2]  = 0x02
RC[3]  = 0x04
RC[4]  = 0x08
RC[5]  = 0x10
RC[6]  = 0x20
RC[7]  = 0x40
RC[8]  = 0x80
RC[9]  = 0x1B
RC[10] = 0x36
```



Decryption

- Unlike DES, AES is not a Feistel network; Decryption is an entirely different function.
- All layers must be inverted:
  - ShiftRows Layer $\longrightarrow$ Inv ShiftRows Layer
  - MixColumn Layer $\longrightarrow$ Inv MixColumn Layer
  - Byte Substitution Layer $\longrightarrow$ Inv Byte Substitution Layer
  - The Key Addition Layer is its own inverse
- Decryption key schedule:
  - Subkeys are needed in reversed order
  - In practice, the same key schedule is used and all the subkeys are precomputed.

Decryption

Inv ShiftRows Layer:

$$\begin{bmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_5 & B_9 & B_{13} & B_1 \\ B_{10} & B_{14} & B_2 & B_6 \\ B_{15} & B_3 & B_7 & B_{11} \end{bmatrix} \Longrightarrow \begin{bmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_1 & B_5 & B_9 & B_{13} \\ B_2 & B_6 & B_{10} & B_{14} \\ B_3 & B_7 & B_{11} & B_{15} \end{bmatrix}$$

- Row 1: no shift
- Row 2: one positions right shift
- Row 3: two positions right shift
- Row 4: three positions right shift

Decryption

Inv MixColumn Layer:

- The inverse of the coefficient matrix is used

$$A = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \quad \Longrightarrow \quad A^{-1} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}$$

- The output is now computed as

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \times \overbrace{\begin{bmatrix} C_0 & C_4 & C_8 & C_{12} \\ C_1 & C_5 & C_9 & C_{13} \\ C_2 & C_6 & C_{10} & C_{14} \\ C_3 & C_7 & C_{11} & C_{15} \end{bmatrix}}^{\text{Input}} = \overbrace{\begin{bmatrix} B_0 & B_4 & B_8 & B_{12} \\ B_5 & B_9 & B_{13} & B_1 \\ B_{10} & B_{14} & B_2 & B_6 \\ B_{15} & B_3 & B_7 & B_{11} \end{bmatrix}}^{\text{Output}}$$

Decryption (Inv Byte Substitution Layer)

Inverse AES S-box, input byte = xy

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Implementation

- Compared to DES, AES has a software friendly design.
- AES is well suited for both low and high powered devices
  - Straightforward implementation fo restriceted devices (8-bit processors) that are used smart cards, low-cost IoT devices, etc.
  - For 32- and 54-bit CPUs, the implementation is entirely Table-based.
- A typical software implementaion of AES can reach a throughput of 1.6 Gbit/s
- Special hardware instructions can increase the throughput to beyond 16 Gbit/s

Security

- Brute-force attacks
  - Not feasible
- Side-channel attacks
  - More successful, but usually require extra information or direct access to hardware
  - Example: There has been attacks on hardware implementations using differential fault analysis that allows recovery of a key using $\approx 2^{32}$ operatoins.
- Quantum attacks
  - AES-128 and AES-192 are not considered quantum resistant due to their smaller key sizes

COMPSCI 4CR3 - Applied Cryptography

Jake Doliskani

McMaster University

Hash Functions

This lecture

- Why we need hash functions
- Important properties of hash functions
- The birthday paradox
- An overview of different families of hash functions
- SHA-2

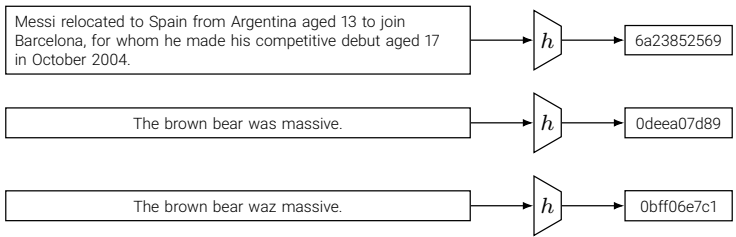
The need for hash functions

We want to be able to represent arbitrarily large data using fixed-size values.

- Extremely important, computationally.
  - Used for hash tables to index large strings
  - Used in many other places, e.g., lossy compression, fingerprints, checksums, error-correcting codes, etc.
- Important for security
  - We will see later in the course how hash functions make digital signatures secure.

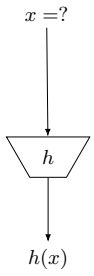
General requirements

1. Should be fast, even for large inputs
2. Should have fixed output size
3. The output should be highly sensitive to all input bits

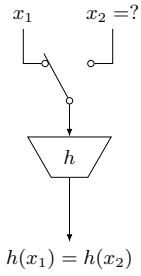


Cryptographic requirements

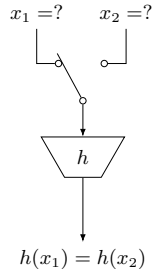
1. Preimage resistance (or one-wayness)
2. Second preimage resistance (or weak collision resistance)
3. Collision resistance (or strong collision resistance)



Preimage resistance



Second preimage resistance



Collision resistance

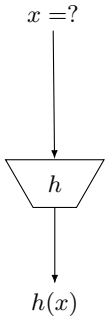
Preimage resistance

A hash function

$$h : \{0,1\}^* \to Y$$

is preimage resistant if given $y \in Y$ in the image of h, it is hard to find $x \in \{0,1\}^*$ such that $h(x) = y$.

Recall: A task T is hard if there is no polynomial time algorithm that can perform T.



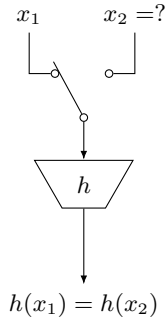
Second preimage resistance

A hash function

$$h : \{0,1\}^* \to Y$$

is second preimage resistant if given $x_1 \in \{0,1\}^*$, it is hard to find $x_2 \in \{0,1\}^*$ such that $x_1 \neq x_2$ and $h(x_1) = h(x_2)$.

Second preimage resistance implies preimage resistance.



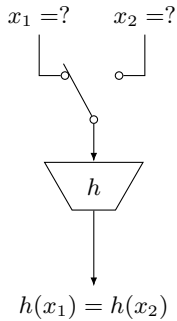
Collision resistance

A hash function

$$h : \{0,1\}^* \to Y$$

is collision resistant if it is hard to find any two $x_1, x_2 \in \{0,1\}^*$ such that $x_1 \neq x_2$ and $h(x_1) = h(x_2)$.

Collision resistance implies second preimage resistance.



Finding collisions

- Suppose the hash function h has an output length of 80 bits.
  - Brute-force: we can try 2^{80} messages to find a collision
  - This can be done by changing spaces, tabs, etc, in actual messages:

```
message1:  Transfer $10 into Trudy's account
message2:  Transfer $10,000 into Trudy,s account
```

- We can do better!
  - The Birthday Attack uses only $\approx 2^{40}$ messages.

The birthday attack

How many people are needed at a party to have a good chance of the same birthday?

$$P(\text{no collision among } t \text{ people}) = \left(1 - \frac{1}{365}\right)\left(1 - \frac{2}{365}\right)\cdots\left(1 - \frac{t-1}{365}\right)$$

For 23 people:

$$\begin{aligned} P(\text{at least one collision}) &= 1 - P(\text{no collision}) \\ &= 1 - \left(1 - \frac{1}{365}\right)\cdots\left(1 - \frac{22}{365}\right) \\ &= 0.507 \end{aligned}$$

The birthday attack

For a hash function with an n-bit output:

$$P(\text{no collision among } t \text{ messages}) = \left(1 - \frac{1}{2^n}\right)\left(1 - \frac{2}{2^n}\right)\cdots\left(1 - \frac{t-1}{2^n}\right) = \prod_{i=1}^{t-1}\left(1 - \frac{i}{2^n}\right)$$

Since $e^{-x} = 1 - x$ for small values of x, we have

$$P(\text{no collision among } t \text{ messages}) \approx \prod_{i=1}^{t-1} e^{-i/2^n} \approx e^{-\frac{t(t-1)}{2^{n+1}}}$$

Set $\lambda = P(\text{at least one collision})$. Then

$$\lambda \approx 1 - e^{-\frac{t(t-1)}{2^{n+1}}}$$

The birthday attack

If we solve for t in

$$\lambda \approx 1 - e^{-\frac{t(t-1)}{2^{n+1}}}$$

we get

$$t \approx 2^{(n+1)/2}\sqrt{-\ln(1-\lambda)}$$

To find a collision (with probability at least 50%) in a hash function with an 80-bit output, we need to try

$$t = 2^{81/2}\sqrt{-\ln(1-0.5)} \approx 2^{40.2}$$

messages.

How to build hash functions

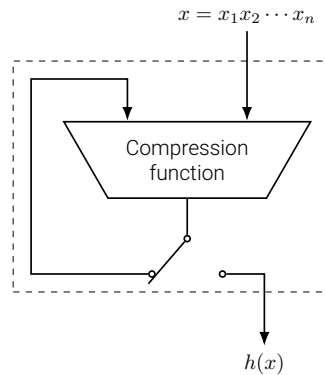
- Dedicated hash functions
  - Algorithms that are specifically designed to serve as hash functions
- Hash functions from Block cipher
  - It is also possible to use block ciphers such as AES to construct hash functions

- Remember, hash functions should be able to hash arbitrary-length messages
  - Use the Merkle–Damgård construction

The Merkle–Damgård construction

1. Segment the message into blocks of equal size
   - Do padding if needed
2. Process each block sequentially using a compression functions
3. Return the output of the last iteration

Example: Secure Hash Algorithms 2 (SHA-2)



Dedicated Hash Functions

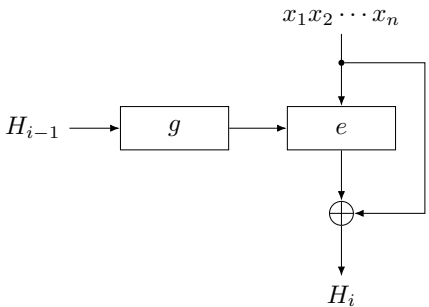
- Hash functions that are custom designed
- Many of these hash functions have been designed in the past 4 decades
- The most popular ones are based on the MD5 family
- MD5 was proposed by Rivest in 1991

- Due to early signs of potential weaknesses in MD5, NIST published SHA
- In 2004, collision-finding attacks against MD5 and SHA-0 where announced by Xiaoyun Wang
- On 23 February 2017, the CWI and Google found collisions in SHA-1

Algorithm	Output	Input	No. of rounds
SHA-224	224	512	64
SHA-256	256	512	64
SHA-384	384	1024	80
SHA-512	512	1024	80

Hash Functions from Block Ciphers

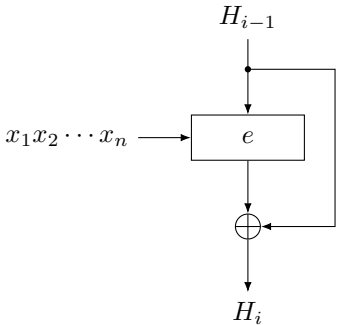
1. Segment the message into blocks of equal size
   - Do padding if needed
2. Feed the blocks into the cipher
3. XOR the output of the cipher with the message block
4. Use the resulting block as the key for the next iteration



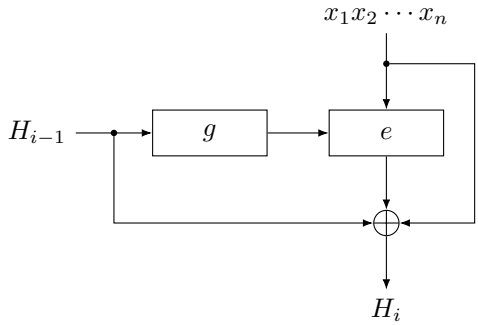
This is called the Matyas–Meyer–Oseas hash function. The equation of the hash function is

$$H_i = e(g(H_{i-1}), x_i) \oplus x_i$$

Other variations



Davies-Meyer



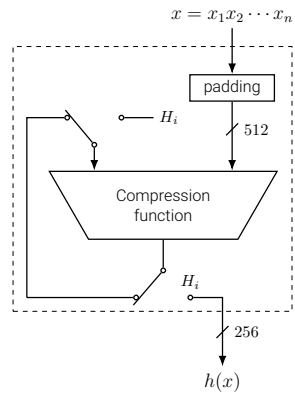
Miyaguchi-Preneel

SHA-2

1. Apply padding so that the message length is a multiple of 512.
2. Divide the message into 512-bit blocks $M_1, M_2, \ldots, M_n$.
3. Sequentially compute

$$H_i = H_{i-1} + C(M_i, H_{i-1})$$

   where C is the is the compression function. The addition is always $\mathrm{mod}\ 2^{32}$.
4. Return H_n



SHA-2: constants

$H_0^{(1)}, H_0^{(2)}, \ldots, H_0^{(8)}$

```
6a09e667 bb67ae85 3c6ef372 a54ff53a 510e527f 9b05688c 1f83d9ab 5be0cd19
```

$K_0, K_1, \ldots, K_{63}$

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

SHA-2: preprocessing

Padding

Suppose the message has length ℓ.

1. Append a "`1`" and k zero bits, where k is the smallest integer such that $\ell + k + 1 \equiv 448 \bmod 512$.
2. Append a 64-bit integer with value equal to ℓ.
3. Example: "`abc`" has length $\ell = 24$. After padding we get

$$\texttt{01100001 01100010 01100011}\ \underbrace{\texttt{100...0}}_{423}\ \underbrace{\texttt{00...011000}}_{64}$$

Dividing

Divide the message into 512-bit blocks $M_1, M_2, \ldots, M_n$

- The first 32 bits of M_i is denoted by $M_i^{(0)}$, the second 32 bits by $M_i^{(1)}$ and so on.

SHA-2: functions

$$
\begin{aligned}
\mathrm{Ch}(x, y, z) &= (x \wedge y) \oplus (x \wedge \neg z) \\
\mathrm{Ma}(x, y, z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\
\Sigma_0(x) &= (x \gg_R 2) \oplus (x \gg_R 13) \oplus (x \gg_R 22) \\
\Sigma_1(x) &= (x \gg_R 6) \oplus (x \gg_R 11) \oplus (x \gg_R 25) \\
\sigma_0(x) &= (x \gg_R 7) \oplus (x \gg_R 18) \oplus (x \gg_S 3) \\
\sigma_1(x) &= (x \gg_R 17) \oplus (x \gg_R 19) \oplus (x \gg_S 10)
\end{aligned}
$$

- Here, $\gg_R$ is right rotation and $\gg_S$ is right shift.
- The inputs x, y, z are 32-bit words.

SHA-2: message schedule

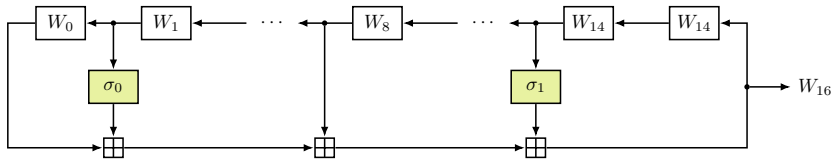
The expanded message blocks $W_0, W_1, ..., W_{63}$ are computed as follows

for $j = 0$ to 15

$$W_j = M_i^{(j)}$$

for $j = 16$ to 63

$$W_j = \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16}$$



SHA-2: compression function

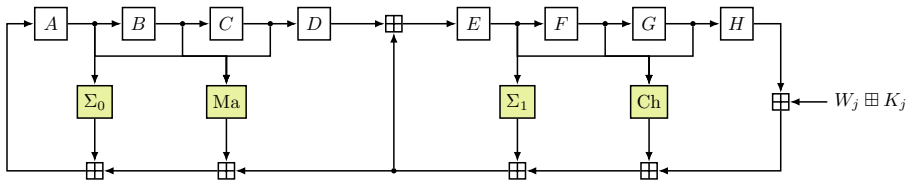
Input: A, B, C, D, E, F, G, H

for $j = 0$ to 63

$T_1 = H + \Sigma_1(E) + \mathrm{Ch}(E, F, G) + K_j + W_j$

$T_2 = \Sigma_0(A) + \mathrm{Ma}(A, B, C)$

$A, B, C, D, E, F, G, H = T_1 + T_2, A, B, C, D + T_1, E, F, G$



SHA-2

1. Pad and divide to get 512-bit blocks $M_1, M_2, \ldots, M_n$
2. for $i = 1$ to n
   - 2.1 $A, B, \cdots, H = H_{i-1}^{(0)}, H_{i-1}^{(1)}, \ldots, H_{i-1}^{(7)}$
   - 2.2 $W_0, W_1 \ldots, W_{63} = \text{schedule}(M_i)$
   - 2.3 $A, B, \cdots, H = \text{compress}(A, B, C, D, E, F, G, H, W)$
   - 2.4 $H_i^{(0)}, H_i^{(1)}, \ldots, H_i^{(7)} = H_{i-1}^{(0)} + A, H_{i-1}^{(1)} + B, \ldots, H_{i-1}^{(7)} + H$
3. Return H_n

COMPSCI 4CR3 - Applied Cryptography

Jake Doliskani

McMaster University

Introduction to Public-key Cryptography

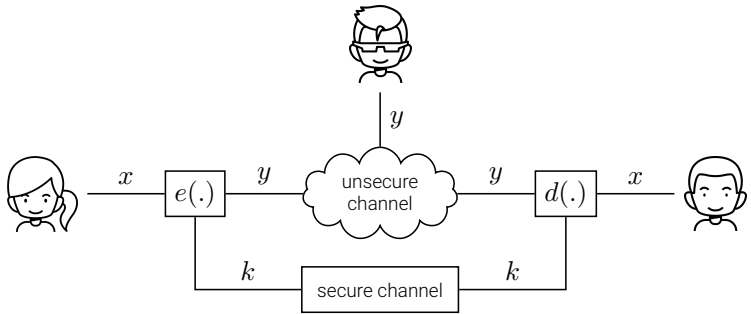
This lecture

- Symmetric cryptography revisited
- Principles of asymmetric cryptography
- Practical aspects of public-key cryptography
- Well-known public-key algorithms
- Some number theory

Symmetric vs. Asymmetric

- Symmetric-key and private-key are the same; Asymmetric-key and public-key are the same. We will use them interchangeably.
- Private-key cryptography has been used for thousands of years. Public-key cryptography was introduced by Diffie, Hellman and Merkle in 1976.
- In private-key cryptography, parties use the same key, while in public-key cryptography, parties use different types of keys.

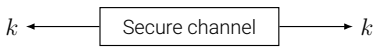
Symmetric-key Cryptography (Revisited)



- The same secret key is used for encryption and decryption
- The encryption and decryption function are very similar

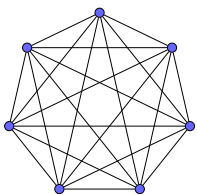
Symmetric-key Cryptography (shortcomings)

- Key distribution:
  There must be a secure channel to transport the key!

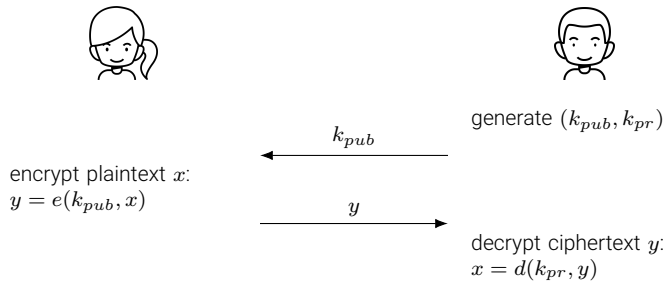


- No protection against a dishonest Alice or Bob: Nonrepudiation

- Number of keys:
  For a network of n users, the number of key pairs can be as large as $n(n-1)/2$



Public-key cryptography



- Bob's key is a pair (k_{pub}, k_{pr}) of public and private keys
- k_{pr} is only known to Bob, while everyone knows k_{pub}
- Alice uses the public key to encrypt, Bob uses the private key to decrypt

A simple key transport

generate (k_{pub}, k_{pr})

k_{pub}

choose a random h
$y = e(k_{pub}, h)$

y

decrypt ciphertext y:
$h = d(k_{pr}, y)$

encrypt message x
$z = \mathrm{AES}(h, x)$

z

decrypt ciphertext z:
$x = \mathrm{AES}^{-1}(h, z)$

How to build public-key systems

One-way function

A function f is one-way if it is easy to compute but hard to invert.

- Easy means computationally feasible: there is a polynomial time algorithm for evaluating f
- Hard means computationally infeasible: there is no polynomial time algorithm for inverting f

Do one-way functions exist?

Security Mechanisms

We can do many things with public-key cryptography:

- Encryption
  - basic functionality
- Key exchange
  - e.g., Diffie–Hellman key exchange
- Nonrepudiation
  - using digital signature algorithms
- Identification
  - using challenge-and-response protocols together with digital signatures

Problem: authentication

The man-in-the-middle attack:

It is possible if public keys are not authenticated.

A solution: certificates

- Certificates bind the identity of a user to their public key

Underlying hard problems

Current/Old

- Integer-Factorization
  - RSA
- Discrete Logarithm
  - Diffie–Hellman key exchange
  - Digital Signature Algorithms
  - Elgamal encryption
- Elliptic Curves DL
  - Diffie–Hellman key exchange (ECDH)
  - Digital Signature Algorithm (ECDSA)

New

- Lattice-based
  - LWE and RLWE constructions,
  - Many others
- Hash-based
  - Digital signatures, e.g., SPHINCS+
- Isogeny-based
  - Digital signatures
  - Encryption
- Code-based
  - Encryption, e.g., Classic McEliece, HQC

Key lengths and security levels

Algorithm Family	Cryptosystems	Security Level (bit)			
		80	128	192	256
Integer factorization	RSA	1024	3072	7680	15360
Discrete logarithm	DH, DSA, Elgamal	1024	3072	7680	15360
Elliptic curves	ECDH, ECDSA	160	256	384	512
Symmetric-key	AES, 3DES	80	128	192	256

The Euclidean Algorithm

Question: how do we compute $\gcd(r_0, r_1)$?

Answer 1: factor r_0, r_1, and collect the greatest common divisor

Example: $\gcd(130, 52) = 26$, since $130 = 2 \cdot 5 \cdot 13$ and $52 = 2^2 \cdot 13$.

- This only works for small numbers because factoring is hard!
- In cryptography, we usually deal with large numbers.

Answer 2: use the Euclidean algorithm

Trick: suppose $r_0 \geq r_1$, then $\gcd(r_0, r_1) = \gcd(r_1, r_0 \bmod r_1)$.

The Euclidean Algorithm

- The identity $\gcd(r_0, r_1) = \gcd(r_1, r_0 \bmod r_1)$ reduces the input size.
- We can repeat until one of the inputs is zero

Example 1: $r_0 = 130, r_1 = 52$
$\gcd(130, 52) = \gcd(52, 26)$
$\gcd(52, 26) = \gcd(26, 0) = 26$

Example 2: $r_0 = 27, r_1 = 21$
$\gcd(27, 21) = \gcd(21, 6)$
$\gcd(21, 6) = \gcd(6, 3)$
$\gcd(6, 3) = \gcd(3, 0) = 3$

The Euclidean Algorithm

Input: positive integers r_0, r_1 such that $r_0 \geq r_1$
Output: $\gcd(r_0, r_1)$

$i = 1$
do
    $i = i + 1$
    $r_i = r_{i-2} \bmod r_{i-1}$
while $r_i \neq 0$
return r_{i-1}

- There are unique numbers s, t such that $\gcd(r_0, r_1) = sr_0 + tr_1$
- With a little change to the above algorithm, we can compute s, t as well

The Extended Euclidean Algorithm

Input: positive integers r_0, r_1 such that $r_0 \geq r_1$
Output: $\gcd(r_0, r_1)$, and integers s, t such that $\gcd(r_0, r_1) = sr_0 + tr_1$

$s_0 = 1, t_0 = 0$
$s_1 = 0, t_1 = 1$
$i = 1$
do

$i = i + 1$
$r_i = r_{i-2} \bmod r_{i-1}$
$q_{i-1} = (r_{i-2} - r_i)/r_{i-1}$
$s_i = s_{i-2} - q_{i-1}s_{i-1}$
$t_i = t_{i-2} - q_{i-1}t_{i-1}$

while $r_i \neq 0$
return $\gcd(r_0, r_1) = r_{i-1}, s = s_{i-1}, t = t_{i-1}$

Euler's phi function

$\varphi(m)$: the number of integers smaller than m that are relatively prime to m

Example:
$\varphi(20) = 8$; the coprime numbers are $1, 3, 7, 9, 11, 13, 17, 19$.
$\varphi(8) = 4$; the coprime numbers are $1, 3, 5, 7$.

Theorem

Suppose m has the prime factorization $m = p_1^{e_1} p_2^{e_2} \cdots p_n^{e_n}$, where the p_i are distinct primes and the e_i are positive integers. Then

$$\varphi(m) = \prod_{i=1}^{n} (p_i^{e_i} - p_i^{e_i - 1}).$$

Euler's theorem

Theorem

Let $a, m > 0$ be integers such that $\gcd(a, m) = 1$. Then

$$a^{\varphi(m)} \equiv 1 \bmod m.$$

Example 1:
$a = 5, m = 21$
$\varphi(21) = (3-1)(7-1) = 12, 5^{12} \equiv 1 \bmod 12$

Example 2:
$a = 12, m = 25$
$\varphi(25) = 5^2 - 5 = 20, 12^{20} \equiv 1 \bmod 25$

COMPSCI 4CR3 - Applied Cryptography

Jake Doliskani

McMaster University

The RSA Cryptosystem

This lecture

- How RSA works
- Implementation aspects
- Finding Large Primes
- Security estimations
- Attacks and Countermeasures

RSA

- Whitfield Diffie and Martin Hellman introduced public-key cryptography in 1976.
- That opened the door to a whole new world
- People were looking for one-way functions to construct encryption algorithms
- Ron Rivest, Adi Shamir, and Leonard Adleman introduced RSA
- RSA became the most widely used public-key scheme
  - Elliptic curve schemes are now very popular

- The underlying one-way function of RSA is the integer factorization problem
  - Multiplying large integers is easy, but factoring large integers is hard

Key generation

1. Choose two large primes p, q
2. Compute $n = pq$ and $\varphi(n) = (p-1)(q-1)$
3. Select the public exponent $e \in \{0, 1, \ldots, \varphi(n) - 1\}$ such that
$$\gcd(e, \varphi(n)) = 1$$
4. Compute the private key d such that
$$de \equiv 1 \bmod \varphi(n)$$
5. Output the private-key, public-key pair (d, e)

Technically, the public key is (n, e).

Encryption, Decryption

Input: public key (n, e) and plaintext x.

1. Compute $y = x^e \bmod n$
2. Return y

Input: Input: private key d and ciphertext y.

1. Compute $x = y^d \bmod n$
2. Return x

Requirements

1. It must be computationally infeasible to find the private key d given the public key (n, e)
2. Key generation, encryption and decryption must all be fast.
   - In particular, it should be easy to do the exponentiations $x^e \bmod n$ and $y^d \bmod n$.
3. Decryption must be an inverse to encryption.
4. For any n there should be many choices of private-key, public-key pairs to avoid a brute-force attack

We can only encrypt at most $\log n$ bits at a time

Correctness

Since $de \equiv 1 \bmod \varphi(n)$, we can write $de = 1 + t\varphi(n)$ for some integer t.
Decrypting the encryption of the message x gives

$$x^{de} = x^{1+t\varphi(n)} = x \cdot \left(x^{\varphi(n)}\right)^t \bmod n$$

Case 1: $\gcd(x, n) = 1$
By Euler's theorem

$$x \cdot \left(x^{\varphi(n)}\right)^t = x \cdot 1 = x \bmod n$$

Case 2: $\gcd(x, n) \neq 1$
Either $x = rp$ or $x = sq$. Without loss of generality assume $x = rp$. Then $\gcd(x, q) = 1$.

Correctness

Since $\gcd(x, q) = 1$, by Euler's theorem

$$\left(x^{\varphi(q)}\right)^t = 1 \bmod n$$

Then

$$\left(x^{\varphi(n)}\right)^t = \left(x^{(p-1)(q-1)}\right)^t = \left(\left(x^{\varphi(q)}\right)^t\right)^{p-1} = 1^{p-1} = 1 \bmod q$$

Using the definition of $\bmod$

$$\left(x^{\varphi(n)}\right)^t = 1 + uq$$

Therefore,

$$x \cdot \left(x^{\varphi(n)}\right)^t = x + xuq = x + (rp)uq = x + nru = x \bmod n. \qquad \square$$

Example (Encryption)

Message $x = 23$

Choose $p = 5, q = 17$
Compute $n = pq = 85$
Compute $\varphi(n) = (5-1)(17-1) = 64$
Choose $e = 9$
Compute $d = e^{-1} = 57 \bmod 64$

$k_{pub} = (85, 9)$

$y = x^e = 23^9 = 28 \bmod 85$

$y = 28$

$y^d = 28^{57} = 23 = x \bmod 85$

Example (Real-world keys)

p = 17528565493044739872307269286412351189332734005147160683201468795448069021563640743291880
840764105927922684458926883355516093643296037472416636393514503276976397639452577007735015775
417403400930807259028198428543307340680583462040539211120968040503177606015248155551862484525
933750113960246788903523850105 51

q = 14812315308169769636190055620708868421825973405717628065958844591753635840891037857676599
981975242493413608211455776408390192069344922052129386535314636903099415714881998723195658534
314647109549986579565457968674884077900741841539121706827409132152859917905057452443328854853
298895409551622034343598655351384 1

e = 65537

d = 827073152544769184892010986664440271894873414915432233250870769289981347099064335511147961
100522415894966019864298010987870276594426020038256040447686179109734969063276749357076084561
545824047955583750600925468766685405309071105696055299143458579876024107332531175496173998293
568329959744488177338553219736944731460688182979764803291311119780236757404479222143763632193
259643112631696857949645051373249798638506272578708123137109326133836014285138872940144313001
437288445453414339914575044842836544312454459298394405824845424266393604409464924872326742284
452380408177274131189867766203329640317927429454566669333073

Exponentiation

How fast can we compute $x^d \bmod n$?

Answer 1: **repeated multiplication**. Requires $d - 1$ multiplications.

Not good enough: $x, d,$ and n are large.

Answer 2: **square and multiply**
Idea: to compute $5^6 \bmod 9$

1. compute 5^2 and $5^4 \bmod 9$
2. compute $5^2 \cdot 5^4 \bmod 9$

three multiplications instead of five

Square-and-Multiply

Let $d = d_k d_{k-1} \cdots d_1 d_0$ be the binary representation of d. Then

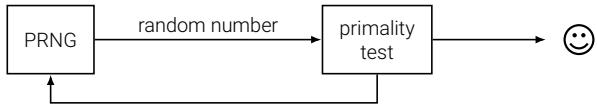
$$x^d = x^{d_k 2^k + d_{k-1} 2^{k-1} + \cdots + 2^1 d_1 + 2^0 d_0} = \left(x^{2^k}\right)^{d_k} \left(x^{2^{k-1}}\right)^{d_{k-1}} \cdots \left(x^{2^1}\right)^{d_1} \left(x^{2^0}\right)^{d_0}.$$

Input: x, d, n
Output: $x^d \bmod n$

```
1 y = x
2 z = 1
3 Let d = d_k d_{k-1} ··· d_1 d_0 be the binary representation of d
4 for i = 0 to k do
5     if d_i = 1 then z = z · y mod n
6     y = y^2 mod n
7 return z
```

▷ $O(\log d)$ multiplications

How to find large primes



For this to be practical

1. We shouldn't have to test too many random numbers to find a prime
2. The primality test should be fast

Primes are common

The Prime Number Theorem

Let $\pi(x) =$ number of primes less that or equal to x. Then

$$\lim_{x \to \infty} \frac{\pi(x)}{x/\ln x} = 1.$$

- The probability of a random number number $\leq x$ being prime is $\approx 1/\ln(x)$.
- For a 512-bit prime number we need to test around

  $$\ln(2^{512}) \approx 355$$

  random numbers.

Primality testing

- Elementary tests
  e.g., try all primes $\leq \sqrt{n}$

- Probabilistic tests
  e.g., Fermat test, Miller–Rabin test

- Deterministic tests
  e.g., Agrawal-Kayal-Saxena test

Theorem

Given an odd integer p, write

$$p - 1 = 2^u r,$$

where r is odd. If we can find an integer a such that

$$a^r \neq 1 \bmod p \quad \text{and} \quad a^{r2^j} = 1 \bmod p$$

for all $j \in \{0, 1, \ldots, u-1\}$, then p is composite. Otherwise, p is probably prime.

Miller-Rabin primality test

Input: odd integer p
Output: "not prime" or "likely prime"

```
1  let p − 1 = 2^u r
2  for i = 1 to s do
3      choose a random a ∈ {2, 3, ..., p − 2}
4      z = a^r mod p
5      if z ≠ 1, −1 mod p then
6          j = 1
7          while j ≤ u − 1 and z ≠ −1 mod p do
8              z = z^2 mod p
9              if z = 1 then return "not prime"
10             j = j + 1
11     if z ≠ −1 mod p then return "not prime"
12 return "likely prime"
```

- s is the number of trials
- The larger the s the more accurate the output
- The output is accurate with probability
$$\approx 1 - 1/4^s$$
- Example: if p is 512-bit number, and we set $s = 30$, then the probability of error is less than $1/2^{60}$.

RSA in practice

For the plain RSA:

- The encryption is deterministic.
  - For a fixed key, a particular plaintext is always mapped to a particular ciphertext.
  - An attacker can sometimes derive partial information or statistical relations between plaintext and ciphertext.
- For the plaintext values $x = 0, 1, -1$, the ciphertext is always one of $y = 0, 1, -1$.
- If both the public exponents e and the plaintexts x are small, the attacker can recover the plaintext.
- The plain RSA is malleable.

RSA in practice

- A scheme is **malleable** if the attacker can change the ciphertext into another ciphertext that corresponds to a known transformation of the plaintext.
- The plain RSA is malleable.
- Example: given a ciphertext y, we can replace it by $t^e y$ where t is some integer. Decryption gives

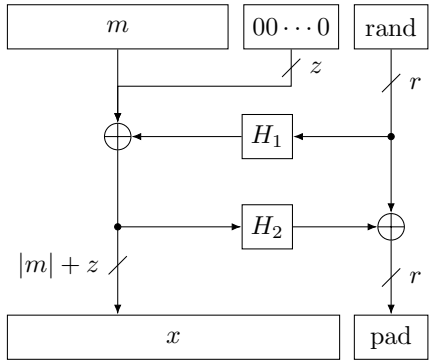
$$(t^e y)^d = t^{ed} x^{ed} = tx \pmod{n}.$$

- Solution: **padding**
- Example padding algorithm: Optimal Asymmetric Encryption Padding (OAEP)

RSA with OAEP

1. Compute $x = (m \| 00 \cdots 0) \oplus H_1(\text{rand})$
2. Compute $\text{pad} = \text{rand} \oplus H_2(x)$
3. Compute $y = (x \| \text{pad})^e \bmod n$

1. Compute $x \| \text{pad} = y^d \bmod n$
2. Recompute $\text{rand} = H_2(x) \oplus \text{pad}$
3. Recompute $m \| 00 \cdots 0 = x \oplus H_1(\text{rand})$
4. Verify that that there are z zeros



Attacks

1. Protocol attacks
2. Mathematical attacks
3. Side-channel attacks

Protocol attacks:

- Exploit weaknesses in the protocols involving RSA
- There has been several such attacks
  - The malleability one we just saw
- To avoid these attacks, follow the guidelines of modern security standards

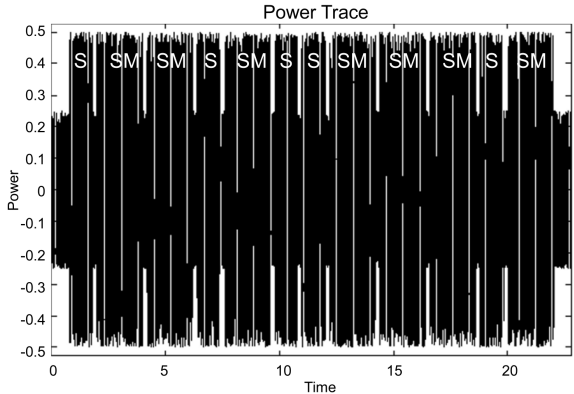
Mathematical Attacks

- The best known attack is factoring the modulus n.
- Factoring n reveals p and q, from which $\varphi(n)$ can be computed. The private key is then $d = e^{-1} \bmod \varphi(n)$.

Some recent RSA factoring records:

Modulus size	Factored on	Factored by
663 bits	2005-05-01	F. Bahr, M. Boehm, J. Franke, and T. Kleinjung
729 bits	2016-05-01	S. Bai, P. Gaudry, A. Kruppa, E. Thomé and P. Zimmermann
762 bits	2018-08-01	Samuel S. Gross.
829 bits	2020-02-01	F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé and P. Zimmermann

Side-Channel Attacks (Example: timing attack)



Operations:	S	SM	SM	S	SM	S	S	SM	SM	SM	S	SM
Private key:	0	1	1	0	1	0	0	1	1	1	0	1

COMPSCI 4CR3 - Applied Cryptography

Jake Doliskani

McMaster University

Public-Key Schemes Based on the Discrete Logarithm Problem

This lecture

- The Diffie-Hellman key exchange
- Cyclic groups
- The discrete logarithm problem
- Security of the Diffie-Hellman Key Exchange
- The Elgamal encryption scheme

The Diffie-Hellman Key Exchange

- Proposed by Whitfield Diffie and Martin Hellman in 1976
- The first asymmetric scheme published in the open literature
- Widely used, e.g., SSH, TLS, IPSec.

Set-up:

1. Choose a large prime p
2. Choose $\alpha \in \{2, 3, \ldots, p-2\}$
3. Publish p, α

The Diffie-Hellman Key Exchange

prime p,
$\alpha \in \mathbb{Z}_p$

Choose $a \in \{2, 3, \ldots, p-2\}$
Compute $k_A = \alpha^a \pmod{p}$

Choose $b \in \{2, 3, \ldots, p-2\}$
Compute $k_B = \alpha^b \pmod{p}$

k_A →

← k_B

Compute $k_{AB} = k_B^a \pmod{p}$

Compute $k_{AB} = k_A^b \pmod{p}$

The shared key is k_{AB}

The DH Key Exchange (example)

$p = 31, \alpha = 19$

Choose $a = 8$
Compute $k_A = 19^8 = 9 \pmod{31}$

Choose $b = 12$
Compute $k_B = 19^{21} = 4 \pmod{31}$

9

4

Compute $k_{AB} = 4^8 = 2 \pmod{31}$

Compute $k_{AB} = 9^{12} = 2 \pmod{31}$

The shared key is $2 \in \mathbb{Z}_{31}$

Groups

A group G is a set of elements equipped with a binary operation $*$ that satisfies the following properties:

1. Closure: for every $a, b \in G$ it holds that $a * b \in G$.
2. Associativity: $a, b, c \in G$ it holds that $(a * b) * c = a * (b * c)$.
3. Neutral element: there exists an element $e \in G$ such that $a * e = e * a = a$ for all $a \in G$.
4. For every $a \in G$, there exists an element $a \in G$ such that $a * b = b * a = e$; b is called the inverse of a and is denoted by a^{-1}.

A group G is called abelian if $a * b = b * a$ for all $a, b \in G$.

Groups (examples)

- $(\mathbb{Z}, +)$: The group of integers $\{\dots, -2, -1, 0, 1, 2, \dots\}$ with the usual addition operations. The neutral element is $e = 0$, and $-a$ is the inverse of a.
- $(\mathbb{C}^\times, \times)$: The set of nonzero complex numbers under multiplication. The identity element is $e = 1$.
- The set of invertible 2×2 matrices over the real numbers under matrix multiplication:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \text{ such that } ad - bc \neq 0, \quad e = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad a, b, c, d \in \mathbb{R}$$

The first two groups are abelian, but the last one is not.

Groups

Let $\mathbb{Z}_n^\times$ be the set of all integers $x \in \{1, 2, \ldots, n-1\}$ that are coprime to n, i.e., $\gcd(x, n) = 1$. Then $\mathbb{Z}_n^\times$ is an abelian group under multiplication modulo n. The identity element is $e = 1$.

Example: multiplication table for $\mathbb{Z}_8^\times$

$\times$	1	3	5	7
1	1	3	5	7
3	3	1	7	5
5	5	7	1	3
7	7	5	3	1

Cyclic groups

A group G is finite if it has a finite number of elements. We denote by $|G|$ the cardinality (or the order) of G.

Example: The order of $\mathbb{Z}_n^\times$ is $\varphi(n)$. So, $\mathbb{Z}_8^\times$ has $\varphi(8) = 4$ elements.

The order $\mathrm{ord}(a)$ of an element $a \in G$ is the smallest integer $k \geq 1$ such that

$$a^k = \underbrace{a * a * \cdots * a}_{k \text{ times}} = 1.$$

Example: The order of $5 \in \mathbb{Z}_8^\times$ is 2.

Cyclic groups

A group G that contains an element α with order $\mathrm{ord}(\alpha) = |G|$ is called cyclic. In this case, α is called a primitive element (or a generator).

Example: The group $\mathbb{Z}_{11}^{\times}$ is cyclic:

- $|\mathbb{Z}_{11}^{\times}| = 10$
- $\mathrm{ord}(2) = 10$, so, 2 is a primitive element.

Theorem

For every prime p, the group $\mathbb{Z}_p^{\times}$ us cyclic.

Order of elements

Theorem

Let G be a finite group. For every $a \in G$

1. $a^{|G|} = 1$,
2. $\mathrm{ord}(a)$ divides $|G|$.

Example: $\mathbb{Z}_{11}^{\times}$

$$\begin{aligned} \mathrm{ord}(1) &= 1, & \mathrm{ord}(6) &= 10, \\ \mathrm{ord}(2) &= 10, & \mathrm{ord}(7) &= 10, \\ \mathrm{ord}(3) &= 5, & \mathrm{ord}(8) &= 10, \\ \mathrm{ord}(4) &= 5, & \mathrm{ord}(9) &= 5, \\ \mathrm{ord}(5) &= 5, & \mathrm{ord}(10) &= 2. \end{aligned}$$

Order of elements

Theorem

Let G be a finite cyclic group. Then

1. The number of primitive elements of G is $\varphi(|G|)$,
2. If $|G|$ is prime, then all elements $a \neq 1$ in G are primitive.

Example 1: $\mathbb{Z}_{11}^{\times}$

- $\varphi(|G|) = \varphi(10) = 4$
- Primitive elements: $2, 6, 7, 8$.

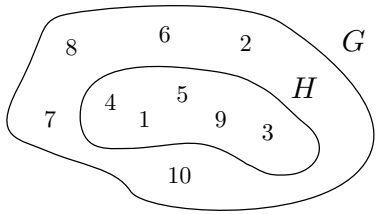
Example 2: the group $H = \{1, 3, 4, 5, 9\}$ with multiplication modulo 11

- $\varphi(|H|) = \varphi(5) = 4$
- Primitive elements: $3, 4, 5, 9$.

Subgroups

A subgroup H of a group G is a subset of G that is itself a group.

Example: the subgroup $H = \{1, 3, 4, 5, 9\}$ of $\mathbb{Z}_{11}^{\times}$.
Multiplication is done modulo 11, the same as in $\mathbb{Z}_{11}^{\times}$



Lagrange's Theorem

For every subgroup H of a group G, $|H|$ divides $|G|$.

Subgroups

Let G be a cyclic group of order n with generator α. Then

- For every integer k that divides n there is exactly one cyclic subgroup $H \leq G$ of order k.
- The subgroup H is generated by $\alpha^{n/k}$
- There are exactly k element $a \in G$ that satisfy $a^k = 1$.

Example: $\mathbb{Z}_{11}^{\times}$

- has order $n = 10$, and is generated by $\alpha = 8$.
- There is exactly one subgroup of order 2 generated by

$$\alpha^{n/k} = 8^{10/2} = 32768 = 10 \pmod{11}$$

The Discrete Logarithm Problem (DLP)

DLP in $\mathbb{Z}_p^\times$:

Let $\alpha \in \mathbb{Z}_p^\times$ be a generator. Given any $\beta \in \mathbb{Z}_p^\times$, the DLP is the problem of finding an integer $1 \leq x \leq p-1$ such that

$$\alpha^x = \beta \pmod{p}.$$

- The integer x is called the discrete logarithm of β to the base α.
- We write $x = \log_\alpha \beta \pmod{p}$
- Example: in $\mathbb{Z}_{47}^\times$: $\log_5 13 = 11 \pmod{47}$, and $\log_2 36 = 17 \pmod{47}$

Generalized DLP

DLP in any cyclic group:

Let G be a cyclic group of order n and let $\alpha \in G$ be a generator. Given any $\beta \in G$, the DLP is the problem of finding an integer $1 \leq x \leq n$ such that

$$\alpha^x = \beta.$$

- Here $\alpha^x = \alpha * \cdots * \alpha$ (x times)
- Example: in $\mathbb{Z}_{47}^{\times}$: $\log_5 13 = 11 \pmod{47}$, and $\log_2 36 = 17 \pmod{47}$

Is DLP hard in all groups?

- $\mathbb{Z}_p^+$ is cyclic of order p.
- The operation is normal addition $\bmod p$.
- A generator $\alpha \in \mathbb{Z}_p^+$ is an element such that every $\beta \in \mathbb{Z}_p^+$ is a repeated sum of α.
- DLP: given $\beta \in \mathbb{Z}_p^+$, find $1 \leq x \leq p-1$ such that

$$x\alpha = \underbrace{\alpha + \cdots + \alpha}_{x \text{ times}} = \beta.$$

Solution: compute $x = \alpha^{-1}\beta$
Computing inverses $\bmod p$ is easy.

Is DLP hard in all groups?

(Hard) DLP groups that have been proposed for cryptography:

- The multiplicative group $\mathbb{Z}_p^\times$
  - Classical DHKE, Elgamal encryption, the Digital Signature Algorithm
- The cyclic group formed by an Elliptic Curve.
- The multiplicative subgroups of the Galois Field $\mathrm{Gal}(2^n)$
  - Not as popular as $\mathbb{Z}_p^\times$, because attacks against them are more efficient
- Hyperelliptic Curves or algebraic varieties
  - Generalization of elliptic curves

Attacks against DLP

- Generic algorithms
  - Brute-Force Search
  - Shanks' Baby-Step Giant-Step Method
  - Pollard's Rho Method
  - Pohlig-Hellman Algorithm
- Nongeneric algorithms
  - The Index-Calculus Method

Brute-Force Search: try all values of $1 \leq x \leq n$ until you find an x such that

$$\alpha^x = \beta$$

Shanks' Baby-Step Giant-Step method

Let $n = |G|$ and $m = \lfloor \sqrt{n} \rfloor$. To find x such that $\alpha^x = \beta$, we write

$$x = x_g m + x_b, \quad \text{for } 0 \leq x_g, x_b < m.$$

Then

$$\beta = \alpha^{x_g m + x_b} \Rightarrow \beta \cdot (\alpha^{-m})^{x_g} = \alpha^{x_b}$$

1. Compute $\gamma = \alpha^{-m}$
2. Compute all the values γ^i for $i = 0, 1, \ldots, m-1$, and store them. (Giant step)
3. For each value $0 \leq x_b < m$ check if there is an i such that

$$\beta^{-1}\alpha^{x_b} = \gamma^i \quad \text{(Baby step)}$$

Complexity: $O(\sqrt{n})$ time, and $O(\sqrt{n})$ memory

Pollard's Rho method

1. Let $n = |G|$. Consider the sequence $\{x_i\}$ given by $x_i = \alpha^{a_i}\beta^{b_i}$, where the pairs (a_i, b_i) are computed in a way that the sequence "looks random".
2. Use a cycle finding algorithm to find (a, b) and (c, d) such that
$$\alpha^a \beta^b = \alpha^c \beta^d.$$
3. Substituting $\beta = \alpha^x$ gives $a + bx = c + dx \pmod{n}$.
4. The discrete logarithm is
$$x = \frac{a - c}{d - b} \pmod{n}$$

- Complexity: $O(\sqrt{n})$ time, and $O(1)$ memory
- Much better than Shanks' Baby-Step Giant-Step Method

Pohlig-Hellman algorithm

1. Let $n = |G|$. Factor n into prime factors: $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$.
2. Compute the discrete logarithm in the subgroups $G_i \leq G$ of size $|G_i| = p_i^{e_i}$.
3. Use the Chinese Remainder Theorem to recover the discrete logarithm in G.

- Efficient only when the prime factors p_i are not too large.
- The discrete logarithm in the G_i can be computed using the Pollard's Rho Method.
- Complexity: $O(\sum_{i=1}^{k}(\log n + \sqrt{p_i})e_i)$

The Index-Calculus method

- Nongeneric algorithm, i.e., works for specific groups.
- Has subexponential running time for the groups $\mathbb{Z}_p^\times$ and $\mathrm{Gal}(2^m)^\times$
- Idea: use the property that a non-negligible fraction of the elements of G can be expressed as products of elements of a small subset of G.
- Using this subset we can collect some linear relations and solve a linear system of equations.

Complexity: $L_n[1/2, \sqrt{2} + o(1)]$, where L_n refers to the L-notation.

Better algorithms: Number Field Sieve, Function Field Sieve

Security of Diffie-Hellman Key Exchange

- Active attaks: the basic version of DHKE is not secure against MITM
- Passive attacks: the security is based on the Diffie–Hellman Problem

The Diffie-Hellman Problem (DHP): Let G be a finite cyclic group and let $\alpha \in G$ be a generator. Given α^a and α^b for some unknown integers a, b, compute α^{ab}.

- If Trudy knows how to solve DLP, then he can solve DHP.
- In general, we don't know if DLP and DHP are equivalent.

The Elgamal encryption scheme

Public parameters:

- large prime p
- generator $\alpha \in \mathbb{Z}_p^\times$

Choose $b \in \{2, 3, \ldots, p-2\}$
Compute $k_B = \alpha^b \pmod p$

k_B

Choose $a \in \{2, 3, \ldots, p-2\}$
Compute $k_A = \alpha^a \pmod p$
Compute $k_{AB} = k_B^a \pmod p$
Encrypt plaintext $x \in \mathbb{Z}_p^\times$:
$y = x \cdot k_{AB} \pmod p$

(k_A, y)

Compute $k_{AB} = k_A^b \pmod p$
Decrypt ciphertext $y \in \mathbb{Z}_p^\times$:
$x = y \cdot k_{AB}^{-1} \pmod p$

The Elgamal encryption scheme

Public parameters:

- prime 53
- generator $27 \in \mathbb{Z}_{53}^{\times}$

Choose $12 \in \{2, 3, \ldots, 51\}$
Compute $k_B = 27^{12} = 46 \pmod{53}$

46 ←

Choose $32 \in \{2, 3, \ldots, 51\}$
Compute $k_A = 27^{32} = 24 \pmod{53}$
Compute $k_{AB} = 46^{32} = 42 \pmod{53}$
Encrypt plaintext $21 \in \mathbb{Z}_{53}^{\times}$:
$34 = 21 \cdot 42 \pmod{53}$

→ $(24, 34)$

Compute $k_{AB} = 24^{12} = 42 \pmod{53}$
Decrypt ciphertext $34 \in \mathbb{Z}_{53}^{\times}$:
$21 = 34 \cdot 42^{-1} \pmod{53}$

Security (passive attacks)

- Security relies on the Diffie-Hellman problem
- The only known attack is through solving DLP

1. Find Bob's secrete key by solving DLP:
$$b = \log_{\alpha} k_B \pmod{p}$$
2. Compute the shared key using Alice's k_A
$$k_{AB} = k_A^b \pmod{p}$$
3. Recover the message:
$$x = y \cdot k_{AB}^{-1} \pmod{p}$$

Security (active attacks)

- MITM (like any other public-key scheme), public keys should be authenticated
- Alice's secrete exponent should not be reused.

1. If Alice reuses the exponent a, then there are two ciphertexts $(y_1, k_A), (y_2, k_A)$ over the channel.
2. If Trudy knows the first message x_1, he can compute

$$k_{AB} = y_1 x_1^{-1} \pmod{p}$$

- Like plain RSA, plain Elgamal is malleable

1. Trudy can replace (k_A, y) with (k_A, sy).
2. Bob decrypts $sy \cdot k_{AB}^{-1} = s \cdot (x \cdot k_{AB}) \cdot k_{AB}^{-1} = sx \pmod{p}$

COMPSCI 4CR3 - Applied Cryptography

Jake Doliskani

McMaster University

Digital Signatures

This lecture

- The principle of digital signatures
- Security services
- The RSA digital signature scheme
- The Elgamal digital signature scheme

Digital signature

A digital signature scheme consists of three algorithms:

- Gen: generates a key pair (k_{pr}, k_{pub}).
- Sig: takes a private key k_{pr} and a message x. Outputs a signature s.
- Ver: takes a public key k_{pub}, a signature s and a message x. Outputs a bit $b \in \{0, 1\}$.

For a signature scheme to be practical, all these algorithms must be efficient.

Digital signature

generate keys: $(k_{pr}, k_{pub}) \leftarrow \mathsf{Gen}$
publish the public key k_{pub}

k_{pub}

sign message: $s = \mathsf{Sig}(k_{pr}, x)$

x, s

verify signature:
$b = \mathsf{Ver}(x, s, k_{pub})$
accept if $b = 1$; reject if $b = 0$

Security Services (core)

1. Confidentiality
   - Information is kept secret from all but authorized parties.
2. Integrity
   - Messages have not been modified in transit.
3. Message Authentication
   - The sender of a message is authentic. An alternative term is data origin authentication.
4. Nonrepudiation
   - The sender of a message can not deny the creation of the message.

Security Services (other)

5. Identification
   - Establish and verify the identity of an entity, e.g., a person, a computer or a credit card.
6. Access control
   - Restrict access to the resources to privileged entities.
7. Availability
   - Assures that the electronic system is reliably available.
8. Auditing
   - Provide evidence about security-relevant activities, e.g., by keeping logs about certain events.

The RSA signature scheme

generate $d, (n, e)$
publish the public key (n, e)

(n, e)

sign message: $s = x^d \pmod{n}$

x, s

verify signature:
$y = s^e \pmod{n}$
$b = (y \stackrel{?}{=} x)$
accept if $b = 1$; reject if $b = 0$

Proof of correctness:

$$s^e = (x^d)^e = x^{de} = x \pmod{n}$$

The RSA signature scheme (example)

choose $p = 5, q = 17, n = pq = 85$
compute $\phi(n) = (5-1)(17-1) = 64$
choose $e = 9$
compute $d = e^{-1} = 57 \pmod{64}$

$(85, 9)$ ←

message $x = 6$
sign: $s = 6^{57} = 11 \pmod{85}$

$x,\ s$ ←
$6, 11$

verify signature:
$y = 11^9 = 6 \pmod{85}$
$b = (6 \stackrel{?}{=} 6) = 1$
accept.

Security

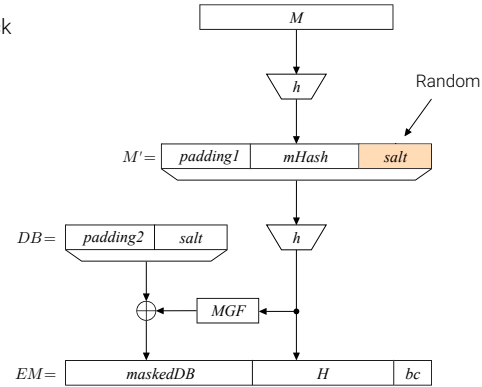
- Algorithmic attacks
  - attack the underlying RSA scheme by computing the private key d
- Existential Forgery
  - generate a valid signature for some message x

Existential Forgery: work backwards to generate a valid signature!

1. Choose a signature $s \in \mathbb{Z}_n$
2. Compute the message $x = s^e \pmod{n}$
3. The signature is (x, s)
4. The signature is valid since $x = s^e \pmod{n}$

RSA padding: The Probabilistic Signature Standard

- Pad the plain RSA to prevent the the existential forgery attack
- The padding method is in fact an encoding method called Encoding Method for Signature with Appendix (EMSA)
- EMSA is probabilistic:
  - Signing a message twice gives different (random) signatures!



The signature: $s = EM^d \pmod{n}$

The Elgamal signature scheme

Key generation:

1. Choose a large prime p
2. Choose a generator $\alpha \in \mathbb{Z}_p^\times$
   - We can also use a generator $\alpha \in G$ for some subgroup $G \leq \mathbb{Z}_p^\times$
3. Choose a random integer $d \in \{2, 3, \ldots, p-2\}$
4. Compute $\beta = \alpha^d \pmod{p}$

- Public parameters: (p, α)
- Public key: β
- Private key: d

The Elgamal signature scheme

Signature generation:
Input: x, d

1. Choose a random ephemeral key $k_E \in \{0, 1, \cdots, p-2\}$ such that $\gcd(k_E, p-1) = 1$
2. Compute $r = \alpha^{k_E} \pmod{p}$ and $s = (x - d \cdot r) k_E^{-1} \pmod{p-1}$
3. Return (r, s)

Signature verification:
Input: $x, (r, s)$

1. Compute $t = \beta^r \cdot r^s \pmod{p}$
2. Return "invalid" if $t \neq \alpha^x \pmod{p}$; otherwise return "valid"

Correctness

If we rewrite $s = (x - d \cdot r)k_E^{-1} \pmod{p-1}$, we get $x = d \cdot r + k_E s \pmod{p-1}$

So, by Fermat's little theorem

$$\alpha^x = \alpha^{d \cdot r + k_E s} \pmod{p}.$$

On the other hand

$$\begin{aligned} \beta^r r^s &= (\alpha^d)^r (\alpha^{k_E})^s \pmod{p} \\ &= \alpha^{d \cdot r + k_E s} \pmod{p}. \end{aligned}$$

Therefore, if (r, s) is a valid signature,

$$\alpha^x = \beta^r r^s \pmod{p}. \qquad \square$$

The Elgamal signature scheme (example)

Public parameters:

- prime $p = 53$
- generator $27 \in \mathbb{Z}_{53}^{\times}$

choose $d = 25$
compute $\beta = \alpha^d = 51 \pmod{53}$

51 ←

message: $x = 41$
choose $k_E = 19$
compute r, s:
$r = \alpha^{k_E} = 31 \pmod{53}$
$s = (x - d \cdot r)k_E^{-1} = 38 \pmod{52}$

$x,\ r,\ s$ ←
$41, 31, 38$

verify signature:
$t = \beta^r r^s = 34 \pmod{53}$
$\alpha^x = 34 \pmod{53}$
$t = \alpha^x \Longrightarrow$ valid signature.

Security

- Computing Discrete Logarithms
- Reuse of the Ephemeral Key
- Existential Forgery Attack

Computing DLP:

- Trudy can obtain d, k_E from $\beta = \alpha^d$ and $r = \alpha^{k_E} \pmod{p}$
- He can sign arbitrary messages

Security

Reuse of the Ephemeral Key:

$$s_1 = (x_1 - d \cdot r)k_E^{-1} \pmod{p-1}$$
$$s_2 = (x_2 - d \cdot r)k_E^{-1} \pmod{p-1}$$
$$\Downarrow$$
$$s_1 k_E + d \cdot r = x_1 \pmod{p-1}$$
$$s_2 k_E + d \cdot r = x_2 \pmod{p-1}$$

May have multiple solutions, Trudy has to find the correct one.

Trudy can compute d, k_E, and sign arbitrary messages.

Security

Existential Forgery Attack:

1. Select integers i, j such that $\gcd(j, p-1) = 1$
2. Compute $r = \alpha^i \beta^j \pmod{p}$ and $s = -rj^{-1} \pmod{p-1}$
3. Compute $x = si \pmod{p-1}$
4. The message and signature are $x, (r, s)$

Countermeasure: hash the message:
$s = (h(x) - d \cdot r)k_E^{-1} \pmod{p-1}$

Verification:

$$
\begin{aligned}
t &= \beta^r r^s \pmod{p} \\
&= \alpha^{dr} \alpha^{(i+jd)s} \pmod{p} \\
&= \alpha^{dr} \alpha^{(i+jd)(-rj^{-1})} \pmod{p} \\
&= \alpha^{dr-dr} \alpha^{-rij^{-1}} \pmod{p} \\
&= \alpha^{si} \pmod{p} \\
&= \alpha^{x} \pmod{p}
\end{aligned}
$$

The Digital Signature Algorithm (DSA)

Key Generation:

1. Generate a prime $2^{1023} < p < 2^{1024}$
2. Find a prime divisor q of $p-1$ such that $2^{159} < q < 2^{160}$
3. Find an element $\alpha \in \mathbb{Z}_p^\times$ of order q
4. Choose a random integer $0 < d < q$
5. Compute $\beta = \alpha^d \pmod{p}$

Other options:

p	q	signature
1024	160	320
2048	224	448
3072	256	512

- Public parameters: (p, q, α)
- Public key: β
- Private key: d

The DSA

Signature generation:
Input: x, d

1. Choose a random ephemeral key $k_E \in \{0, 1, \cdots, q-1\}$
2. Compute $r = (\alpha^{k_E} \bmod p) \pmod q$ and $s = (h(x) + d \cdot r)k_E^{-1} \pmod q$
3. Return r, s

Signature verification:
Input: x, r, s

1. Compute $w = s^{-1} \pmod q$, $u_1 = wh(x) \pmod q$, and $u_2 = wr \pmod q$
2. Compute $t = (\alpha^{u_1}\beta^{u_2} \bmod p) \pmod q$
3. Return "valid" if $t = r$; otherwise return "invalid"

Correctness

$$
\begin{aligned}
s = (h(x) + d \cdot r)k_E^{-1} \bmod q \quad &\Longrightarrow \quad k_E = s^{-1}h(x) + s^{-1}rd \pmod q \\
&\Longrightarrow \quad k_E = u_1 + u_2 d \pmod q \\
&\Longrightarrow \quad \alpha^{k_E} = \alpha^{u_1 + u_2 d} \pmod p \\
&\Longrightarrow \quad \alpha^{k_E} = \alpha^{u_1}\beta^{u_2} \pmod p \qquad \text{(since } \alpha^d = \beta\text{)} \\
\text{(reduce both sides mod } q\text{)} \quad &\Longrightarrow \quad (\alpha^{k_E} \bmod p) \bmod q = (\alpha^{u_1}\beta^{u_2} \bmod p) \bmod q \\
&\Longrightarrow \quad r = t \qquad \square
\end{aligned}
$$

The DSA (example)

Public parameters:

- primes $p = 53, q = 13$
- generator $\alpha = 10$

choose $d = 8$
compute $\beta = \alpha^d = 24 \pmod{53}$

24 ←

message: $x = 41, h(x) = 6$
choose $k_E = 9$
compute r, s:
$r = (\alpha^{k_E} \bmod p) \bmod q = 2$
$s = (h(x) - d \cdot r)k_E^{-1} = 1 \pmod{13}$

$x,\ r,\ s$ ←
$41, 2, 1$

verify signature:
$w = s^{-1} = 1 \pmod{13}$
$u_1 = w \cdot h(x) = 6 \pmod{13}$
$u_2 = wr = 2 \pmod{13}$
$t = (\alpha^{u_1}\beta^{u_2} \bmod p) \bmod q = 2$
$t = r \Longrightarrow$ valid signature

Prime generation for DSA

Generate primes p, q such that

- $2^{1023} < p < 2^{1024}$ and $2^{159} < q < 2^{160}$,
- $q \mid p - 1$

1 Find a prime $2^{159} < q < 2^{160}$ using the Miller-Rabin algorithm
2 for $i = 1$ to 4096 do
3 Generate a random integer $2^{1023} < M < 2^{1024}$
4 Compute $M_r = M \pmod{2q}$
5 Let $p = M - M_r + 1$
6 If p is prime, then return (p, q)
7 Go to Step 1

- Each iteration of the For-loop selects a random number of the form $p = 2qk + 1$ in the range $(2^{2023}, 2^{1024})$ and tests whether it is a prime.
- Does not take too many trials

COMPSCI 4CR3 - Applied Cryptography

Jake Doliskani

McMaster University

Elliptic Curve DLP

This lecture

- What is an elliptic curve?
- Computing with elliptic curves
- Elliptic curve DLP
- Prorocols based on ECDLP
- Security estimates
- Implementation

Motivation

Problem:

- Systems like RSA and Elgamal require exponetiation in $\mathbb{Z}_p$.
  - When p is large, exponentiation is expensive.
  - Attacks on factorization and DLP are subexponential.

Motivation:
It would be nice to have smaller ring sizes for the same security levels.

Solution:
Use the group of points on an elliptic curve.

Fields

Recall the definition of a ring:

A ring R is a set with the following properties for all $a, b, c \in R$:

- Closure: The results of addition and multiplication are always in R.
- Associativity:

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

- Distributive law: $a \times (b + c) = (a \times b) + (a \times c)$
- Neutral element for addition: $a + 0 = a$
- Neutral element for multiplication: $a \times 1 = a$
- Additive inverse: $a + (-a) = 0$
- Multiplicative inverse: $a \times a^{-1} = 1$ (might not always exists)

Fields

A ring R is called a field if multiplication in R

- is commutative: $a \times b = b \times a$ for all $a, b \in R$
- has inverse: for every $a \in R$ there is an element $b \in R$ such that $a \times b = 1$. We denote the inverse of a by a^{-1}.

Example:

- $\mathbb{Z}_p$, the set of integers mod a prime p. $\mathbb{Z}_p$ is called a finite field.
- $\mathbb{R}, \mathbb{C}$ are infinite fields.

Elliptic Curves

Let K be field. An elliptic curve E over K is a set of points $(x, y) \in K \times K$ that satisfiy the equation

$$y^2 = x^3 + ax + b$$

for some fixed $a, b \in K$ such that

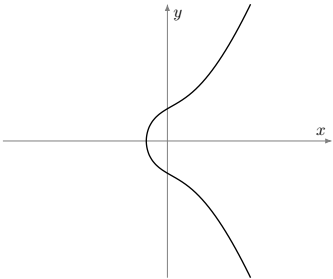
1. $4a^3 + 27b^2 \neq 0$
2. We add a point at infinity, denoted by ∞, to the set of points

Only for certain fields we can write an elliptic curve in the above form. The general equation for an elliptic curve is

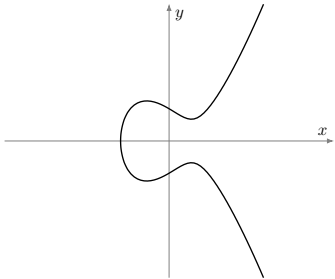
$$E: \ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

Elliptic Curves (over $\mathbb{R}$)

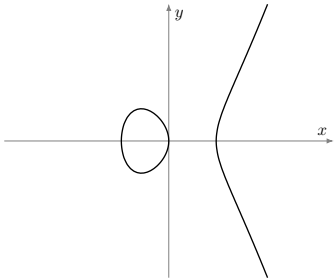
$\mathbb{R}$ is nice! We can draw elliptic curves.



$y^2 = x^3 + 2x + 2$



$y^2 = x^3 - 2x + 2$



$y^2 = x^3 - 3x$

Elliptic Curves (over $\mathbb{Z}_p$)

For cryptography, we use elliptic curves over finite fields.

Let $p \geq 5$ be prime. An elliptic curve E over $\mathbb{Z}_p$ is a set of points $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ that satisfiy the equation

$$y^2 = x^3 + ax + b \pmod{p}$$

for some fixed $a, b \in \mathbb{Z}_p$ such that

1. $4a^3 + 27b^2 \neq 0 \pmod{p}$
2. We add a point at infinity, denoted by ∞, to the set of points

Example: $p = 7$.

$$y^2 = x^3 + x + 1 \pmod{7}.$$

Set of points: $\{(0, 1), (0, 6), (2, 2), (2, 5)\} \cup \{\infty\}$

Group structure

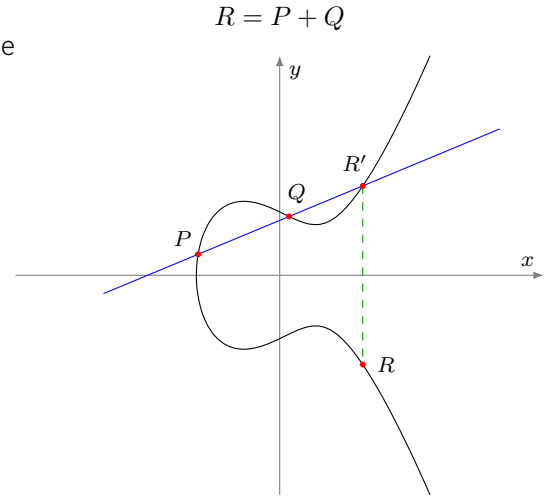
Let $P = (x_1, y_1)$, $Q = (x_2, y_2)$, $R = (x_3, y_3)$ be points on $E: \ y^2 = x^3 + ax + b$. Then

$$x_3 = m^2 - x_1 - x_2,$$
$$y_3 = m(x_1 - x_3) - y_1,$$

where

$$m = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{if } P \neq Q, \\ \dfrac{3x_1^2 + a}{2y_1}, & \text{if } P = Q, \end{cases}$$

If $x_1 = x_2$ and $y_1 = -y_2$, then $P + Q = \infty$.



Point addition (example)

Consider $E : y^2 = x^3 + x + 1 \pmod{13}$. Let $P = (5, 1)$ and $Q = (11, 2)$.

- $[2]P = P + P = (4, 11)$:

$$m = \frac{3x_1^2 + a}{2y_1} = \frac{11}{2} = 12 \pmod{13}.$$
$$x_3 = m^2 - x_1 - x_2 = 4 \pmod{13}$$
$$y_3 = m(x_1 - x_3) - y_1 = 11 \pmod{13}$$

- $P + Q = (1, 4)$:

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{1}{6} = 11 \pmod{13}.$$
$$x_3 = m^2 - x_1 - x_2 = 1 \pmod{13}$$
$$y_3 = m(x_1 - x_3) - y_1 = 4 \pmod{13}$$

The double-and-add algorithm

Let $d = d_k d_{k-1} \cdots d_1 d_0$ be the binary representation of d. Then

$$\begin{aligned}[d]P &= [d_k 2^k + d_{k-1} 2^{k-1} + \cdots + d_1 2^1 + d_0]P \\ &= [d_k]([2^k]P) + [d_{k-1}]([2^{k-1}]p) + \cdots + [d_1]([2]P) + [d_0]P.\end{aligned}$$

Input: point P, integer $d > 0$.
Output: $[d]P$

1 $Q = P$
2 $R = \infty$
3 Let $d = d_k d_{k-1} \cdots d_1 d_0$ be the binary representation of d
4 for $i = 0$ to k do
5 if $d_i = 1$ then $R = R + Q$
6 $Q = [2]Q$
7 return R

▷ $O(\log d)$ additions

Group structure

Theorem

The points on an elliptic curve form an abelian group. Under certain conditions this group is always cyclic.

The group of points of E over a field K is denoted by $E(K)$.

Example:

- $E : y^2 = x^3 + x + 1 \pmod{13}$
- $E(\mathbb{Z}_{13})$ is cyclic of order 18
- Generator: $P = (5, 1)$

$$
\begin{aligned}
[1]\,P &= (5,1) & [10]\,P &= (11,2)\\
[2]\,P &= (4,11) & [11]\,P &= (1,4)\\
[3]\,P &= (0,1) & [12]\,P &= (10,6)\\
[4]\,P &= (8,12) & [13]\,P &= (12,5)\\
[5]\,P &= (12,8) & [14]\,P &= (8,1)\\
[6]\,P &= (10,7) & [15]\,P &= (0,12)\\
[7]\,P &= (1,9) & [16]\,P &= (4,2)\\
[8]\,P &= (11,11) & [17]\,P &= (5,12)\\
[9]\,P &= (7,0) & [18]\,P &= \infty
\end{aligned}
$$

Number of points

Theorem (Hasse)

Let E be an elliptic curve over $\mathbb{Z}_p$. Then

$$p + 1 - 2\sqrt{p} \leq |E(\mathbb{Z}_p)| \leq p + 1 + 2\sqrt{p}.$$

- To have a group of order $\approx 2^n$, we need a prime $p \approx 2^n$.
- We can use point cuonting algorithms to compute $|E(\mathbb{Z}_p)|$.
- The SEA (Schoof–Elkies–Atkin) point counting algorithm has complexity $\tilde{O}((\log p)^4)$

Discrete logarithm problem

Elliptic curve discrete logarithm problem (ECDLP):

Let E be an elliptic curve over $\mathbb{Z}_p$. Let $G \subset E(\mathbb{Z}_p)$ be a cyclic subgroup with $P \in G$ a generator. The discrete logarithm problem in G is as follows:

$$\text{given } T \in G, \text{ find } 0 \leq d < |G| \text{ such that } [d]P = T.$$

- Best known generic algorithm for ECDLP is the Pollard's rho algorithm.
- The complexity of Pollard's rho is $O(\sqrt{p})$.
  - This is the best we can do for many curves

The Diffie-Hellman Key Exchange

Elliptic curve E,
Generator $P \in E$

Choose $a \in \{2, 3, \ldots, |E| - 1\}$
Compute $k_A = [a]P$

Choose $b \in \{2, 3, \ldots, |E| - 1\}$
Compute $k_B = [b]P$

k_A →

← k_B

Compute $k_{AB} = [a]k_B$

Compute $k_{AB} = [b]k_A$

The shared key is k_{AB}

ECDH (example)

$E : y^2 = x^3 + x + 1 \pmod{13}$,
Generator $P = (5, 1) \in E$

Choose $a = 5 \in \{2, 3, \ldots, 17\}$
Compute $k_A = [5]P = (12, 8)$

Choose $b = 12 \in \{2, 3, \ldots, 17\}$
Compute $k_B = [12]P = (10, 6)$

$(12, 8)$ →

← $(10, 6)$

Compute $k_{AB} = [5]k_B = (10, 7)$

Compute $k_{AB} = [12]k_A = (10, 7)$

The shared key is $(10, 7)$

Elliptic Curve Digital Signature Algorithm (ECDSA)

Key generation:

1. Choose a large prime p and an elliptic curve E over $\mathbb{Z}_p$ such that
   - E has a large cyclic subgroup G of prime order q
2. Choose a point $A \in G$ of order q
3. Choose a random positive intgere $d < q$
4. Compute $B = [d]A$

- Public parameters: (p, q, E, A)
- Public key: B
- Private key: d

Elliptic Curve Digital Signature Algorithm (ECDSA)

Signature generation:
Input: d, x

1. Choose an integer as random ephemeral key $0 < k_E < q$
2. Compute $R = [k_E]A = (x_R, y_R)$.
3. Let $r = x_R$ and compute $s = (h(x) + d \cdot r)k_E^{-1} \bmod q$
4. Return r, s

Signature verification:
Input: x, r, s

1. Compute $w = s^{-1} \bmod q$ and $u_1 = wh(x) \bmod q$ and $u_2 = wr \bmod q$
2. Compute $P = [u_1]A + [u_2]B$
3. Return "valid" if $x_P = r$; otherwise return "invalid"

Correctness

$$
\begin{aligned}
s = (h(x) + d \cdot r)k_E^{-1} \bmod q \quad &\implies \quad k_E = s^{-1}h(x) + s^{-1}rd \pmod q \\
&\implies \quad k_E = u_1 + u_2 d \pmod q \\
&\implies \quad [k_E]A = [u_1 + u_2 d]A \\
&\implies \quad [k_E]A = [u_1]A + [u_2 d]A \\
&\implies \quad [k_E]A = [u_1]A + [u_2]B \qquad \text{(since } B = [d]A\text{)} \\
\text{(Since } P = [u_1]A + [u_2]B\text{)} \quad &\implies \quad x_P = x_R = r \qquad \square
\end{aligned}
$$

The ECDSA (example)

Public parameters:

- $p = 29, E : y^2 = x^3 + 2x + 8$
- $q = 29, A = (2, 7)$

choose $d = 8$
compute $B = [d]A = [8](2, 7) = (12, 7)$

$B = (12, 7)$ ←

message: $x = 12, h(x) = 21$
choose $k_E = 17$
compute:
$R = [k_E]A = [17](2, 7) = (27, 5)$,
$r = x_R = 27$,
$s = (h(x) + d \cdot r)k_E^{-1} = 2 \pmod{29}$

$x,\ r,\ s$ ←
$12, 27, 2$

verify signature:
$w = s^{-1} = 15 \pmod{29}$
$(u_1, u_2) = (w \cdot h(x), wr) = (25, 28) \bmod 29$
$P = [u_1]A + [u_2]B = (27, 5)$
$x_R = r \implies$ valid signature

Security

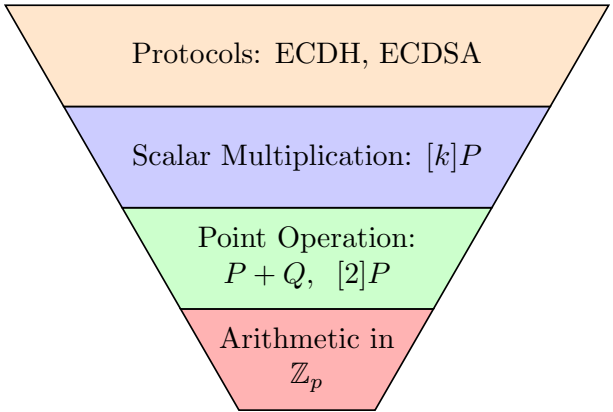
Smaller parameters than RSA or Elgamal:

- There are no sunexponential algorithms, like the ones for integers factorization or finite field DL, for ECDP.
- For the curves chosen in cryptography, the best known algorithm for EDCP is the Pollard's rho algorithm.
- The complexity of Pollard's rho algorithm is $O(\sqrt{p})$.

Practical parameter sizes for elliptic curves:

- For a 160-bit prime p, EDCP provides 80 bits of security
- For a conservative choice of 256-bit prime p, EDCP provide 128 bits of security.

Implementation



Implementation

Software:

- On ARM processors, scalar multiplication can be as fast as a few hundred microseconds.
- On general desktop (x86) processors, scalar multiplication can be as fast as a few tens of microseconds.

Hardware:

- ASIC implementations can be extremely fast, as low as about 10 microseconds.
- Typical embedded (FPGA) implementations achieve a few milliseconds per scalar multiplication.