Slides for COMP 70009 Cryptography Engineering

NOTE: These slides are formatted such that each page has enough space for note taking during studies.

Chapters 7+9 of Nigel Smart's book: symmetric encryption

1. $m \in \mathbb{P}$ set of plaintexts
2. $k \in \mathbb{K}$ set of keys
3. $c \in \mathbb{C}$ set of ciphertext

$e\colon \mathbb{K} \times \mathbb{P} \to \mathbb{C}$ encryption function $c = e_k(m)$
$d\colon \mathbb{K} \times \mathbb{C} \to \mathbb{P}$ decryption function $d_k(c)$ equals m?

1. correctness: $\forall k \in \mathbb{K}\, \forall m \in \mathbb{P}\colon m = d_k(e_k(m))$
2. symmetric encryption: same[1] key used for e and for d
3. secrecy: d and e public (Kerckhoff's Principle); c may be seen to be public, too; secrecy of m rests on secrecy of k (and on secrecy of m, e.g. not passed along in the clear)

[1] "Same" does not necessarily mean "identical" but that these keys are easily derived from each other.

Example: shift cipher

- $\mathbb{P} = \{a, b, \ldots, z\} = \mathbb{C}$
- $\mathbb{K} = \{0, 1, \ldots, 25\}$
- $e_k(m) = m$ shifted by k positions forward in "cyclic alphabet"
- for example, $e_2(a) = c$, $e_3(b) = e$, and $e_2(z) = b$
- extend encryption of letters to words *homomorphically*, e.g. $e_2(hello) = jgnnq$
- Decryption: same as encryption, only revert the order in the cyclic alphabet
- for example, $d_2(b) = z$ and $d_2(jgnnq) = hello$
- cryptosystems often have *weak* keys, for example in the $\mathbb{K}$ above?
- QUESTION: why is this cryptosystem trivial to break?

Example: substitution cipher

- $\mathbb{P} = \{a, b, \ldots, z\} = \mathbb{C}$ as for the shift cipher
- $\mathbb{K} =$ all *permutations* of $\mathbb{P}$
- e_k extended from letters to finite words as for the shift cipher
- for example $e_k(hello) = esvvj$ when permutation k satisfies $k(h) = e$, $k(e) = s$, $k(l) = v$, and $k(o) = j$
- decryption: $d_k(c) = e_{k^{-1}}(c)$ where k^{-1} is *inverse* of permutation k (permutations have group structure)
- therefore: we only need an *encryption device* here for both encryption *and* decryption, since $\mathbb{K}$ has group structure
- $|\mathbb{K}| = 26! \approx 2^{88}$ is a *huge* key space, where $|A|$ is the size of a set A
- QUESTION: is the substitution cipher secure?
- Huge key space alone is no guarantee for security.

Example: Vigenère cipher

- $\mathbb{P} = \{a, b, \ldots, z\}^+ = \mathbb{C}$, where A^+ denotes the set of finite words over alphabet A
- $\mathbb{K} = \prod_{i=1}^{p} Perm(\{a, b, \ldots, z\})$ where $Perm(A)$ denotes the set of permutations of set A
- thus a key k is a tuple $(k_1, k_2, \ldots, k_p)$ of $p > 1$ many permutations of set $\{a, b, \ldots, z\}$
- encryption: $e_k(m_1 m_2 \ldots m_n) = c_1 c_2 \ldots c_n$ where each m_i is in $\{a, b, \ldots, z\}$ and $c_i = k_{[(i-1) \bmod p]+1}(m_i)$
- for example, we have $e_{(k_1,k_2)}(hello) = shljv$ for permutations k_1 and k_2 satisfying $k_1(h) = s$, $k_2(e) = h$, $k_1(l) = l$, $k_2(l) = j$, and $k_1(o) = v$
- note how in this example the two occurrences of l get encrypted into different letters: l by k_1 and j by k_2
- in general, this cipher uses key k_1 on letters $m_1, m_{p+1}, m_{2p+1}, \ldots$ and key k_j on letters m_{qp+j} for all q with $1 \leq qp + j \leq k$

Vigenère cipher decryption

$$d_{(k_1,k_2,\ldots,k_p)} = e_{(k_1^{-1},k_2^{-1},\ldots,k_p^{-1})}$$

- QUESTION: Is the Vigenère cipher more secure than the simple substitution cipher?
- QUESTION: Is the Vigenère cipher secure?
- QUESTION: in what sense are the shift cipher, substitution cipher, and Vigenère cipher *symmetric* encryption?

How can we design symmetric key ciphers that we can *prove* to be secure?

We will study this next by considering three notions of security.

Three notions of security

1. *Information-theoretic security*: also known as *unconditional security*; a cipher cannot be broken even with *infinite* computing power (will formalize what "broken" means later)

2. *Computational Security*: cipher cannot be broken within specified computing power, e.g. where an attacker can compute only within probabilistic polynomial time

3. *Provable security*: show security via a problem reduction; "Cipher can be broken implies that a known and believed to be hard problem can be solved."

QUESTION: Why is the term "provable" in 'provable security' potentially misleading?

Discussion of security notions

- Unconditional security: holds forever
- Computational and provable security: may not hold forever, for example
  - cryptography in a "post-quantum" world
  - hard problems such as integer factorization may turn out to be simple
- Shift cipher, substitution cipher, and Vigenère cipher are all computationally insecure
- Next we study an example of an unconditionally secure cipher, the *one-time pad*

Probability and Ciphers

- think of $\mathbb{P}$, $\mathbb{C}$, and $\mathbb{K}$ as discrete probability distributions
- for example, let $\mathbb{P} = \{a, b, c, d\}$ and $\mathbb{K} = \{k_1, k_2, k_3\}$
- then $p(P = a) = 1/4$ means that the probability that the plaintext letter is 'a' is 0.25
- similarly, $p(K = k_2) = 1/2$ means that the probability that the used key is k_2 equals 0.5
- for each k in $\mathbb{K}$, we set $\mathbb{C}(k) = \{e_k(x) \mid x \in \mathbb{P}\}$
- note that $\mathbb{C}(k)$ is the set of all ciphertexts based on key k
- ASSUMPTION: plaintext and key are *chosen independently*:

$$(9)\quad p(C = c) = \sum_{k|c\in\mathbb{C}(k)} p(K = k) \cdot p(P = d_k(c))$$

Example

$$\mathbb{P} = \{a, b, c, d\} \qquad \mathbb{K} = \{k_1, k_2, k_3\} \qquad \mathbb{C} = \{1, 2, 3, 4\}$$

$e_k(m)$	a	b	c	d
k_1	3	4	2	1
k_2	3	1	4	2
k_3	4	3	1	2

probability distributions for keys and plaintexts in table form:

key	k_1	k_2	k_3
	1/4	1/2	1/4

plaintext	a	b	c	d
	1/4	3/10	3/20	3/10

use (9) to get

$$\begin{aligned} p(C=1) &= p(K=k_1)\cdot p(P=d) + p(K=k_2)\cdot p(P=b) + p(K=k_3)\cdot p(P=c) \\ &= 1/4\cdot 3/10 + 1/2\cdot 3/10 + 1/4\cdot 3/20 \\ &= 0.2625 \end{aligned}$$

That is, the probability that the ciphertext equals 1 is 0.2625.

Example continued

Similarly, we use (9) to compute the entire distribution for ciphertexts:

$$\begin{aligned} P(C=1) &= 0.2625 \qquad (\ldots \text{just computed}) \\ P(C=2) &= 0.2625 \\ P(C=3) &= 0.2625 \\ P(C=4) &= 0.2125 \end{aligned}$$

This distribution is almost uniform for ciphertexts, but ciphertext 4 is *somewhat less likely*. Therefore, the above cryptosystem is *not* unconditionally secure.

To formalize unconditional security, we need to define conditional probabilities next.

Conditional probability

$p(C = c \mid P = m)$ = probability that ciphertext is c given that plaintext is m

Note that we have

$$p(C = c \mid P = m) = \sum_{k|m=d_k(c)} p(K = k)$$

since different keys have disjoint such events. For our previous example:

- we have that $p(C = 1 \mid P = a) = 0$ since no key in $\mathbb{K}$ encrypts a as 1
- we have that $p(C = 3 \mid P = a) = 0.75$ since this is the result of expression $p(K = k_1) + p(K = k_2) = 1/4 + 1/2$

What attacker wants to learn

$p(P = m \mid C = c)$ = probability that plaintext is m, given that ciphertext is c

Compute this from $p(C = c \mid P = m)$ using Bayes' Theorem:

$$p(P = m \mid C = c) = \frac{p(P = m) \cdot p(C = c \mid P = m)}{p(C = c)}$$

ASSUMPTION: for all c in $\mathbb{C}$, we have $p(C = c) > 0$.

QUESTION: why does this assumption not lose generality?

Example computations:

$$p(P = a \mid C = 1) = \frac{p(P = a) \cdot 0}{p(C = 1)} = 0$$

$$p(P = a \mid C = 3) = \frac{p(P = a) \cdot p(C = 3 \mid P = a)}{p(C = 3)} = \frac{(1/4) \cdot 0.75}{0.2625} \approx 0.714$$

Determine most likely plaintext, given ciphertext

For example, let C be 1. Then we compute as in the previous example

$$\begin{aligned} p(P = a \mid C = 1) &= 0 \\ p(P = b \mid C = 1) &= 0.571 \\ p(P = c \mid C = 1) &= 0.143 \\ p(P = d \mid C = 1) &= 0.286 \end{aligned}$$

Therefore, b is the *most likely plaintext* as 0.571 is the maximal probability above.

NOTE: this assumes that we do not have any other contextual information that may inform such probabilities.

We want to prevent inferences about most likely plaintexts: the formal definition of *perfect secrecy* will capture this.

Perfect secrecy

Definition 5.1: A cryptosystem has perfect secrecy if

$$\forall m \in \mathbb{P}\ \forall c \in \mathbb{C}: \qquad p(P = m \mid C = c) = p(P = m)$$

In other words, perfect secrecy implies that we cannot learn anything new about the plaintext once we learn the value of the ciphertext.

Instructive exercises:

- **(Exercise 3 in exercise sheet)** The above definition is equivalent to one based on $P(C = c \mid P = m) = p(C = c)$.
- **(Exercise 4 in exercise sheet)** Let a cryptosystem be perfectly secure. Then it must be the case that $|\mathbb{K}| \geq |\mathbb{C}| \geq |\mathbb{P}|$.

This means that we need as least as many keys as plaintexts for a perfectly secure cryptosystem. Therefore, perfect secrecy has limited practical value.

Shannon's Characterization of Perfect secrecy

Theorem 9.4 (Shannon): A cryptosystem

$$(\mathbb{P}, \mathbb{C}, \mathbb{K}, e_k(\cdot), d_k(\cdot))$$

with

$$|\mathbb{P}| = |\mathbb{C}| = |\mathbb{K}| \tag{1}$$

is perfectly secure iff (meaning "if, and only if"):

(S1) for all k in $\mathbb{K}$, we have that $p(K = k) = 1/|\mathbb{K}|$, and

(S2) for all m in $\mathbb{P}$ and for all c in $\mathbb{C}$, there is a *unique* k in $\mathbb{K}$ with $e_k(m) = c$.

PROOF: We show that perfect secrecy implies the statements in (S1) and (S2) above, under the assumptions on the sizes of the spaces of keys, plaintexts, and ciphertexts in (1) above.

EXERCISE: **(Exercise 5 in exercise sheet)** Complete that proof. That is, show that the statements in (S1) and (S2) imply perfect secrecy under the assumptions in (1).

Proof that perfect secrecy implies (S2)

Without loss of generality we may assume that $p(C = c) > 0$ for all c in $\mathbb{C}$. Now let c in $\mathbb{C}$ and m in $\mathbb{P}$ be fixed. Then $p(C = c) > 0$ and perfect secrecy imply that

$$p(C = c \mid P = m) = p(C = c) > 0 \tag{2}$$

Since that probability is positive, there must be some k in $\mathbb{K}$ with $e_k(m) = c$. It remains to show that this k is unique.

Claim: $|\{e_k(m) \mid k \in \mathbb{K}\}| = |\mathbb{C}|$.

This follows as these two sets are equal: clearly, $\{e_k(m) \mid k \in \mathbb{K}\}$ is a subset of $\mathbb{C}$. To show the converse inclusion we appeal to (2) instantiated with an arbitrary c' in $\mathbb{C}$: so for all all c' in $\mathbb{C}$ there is some k' in $\mathbb{K}$ with $e_{k'}(m) = c'$. Therefore, $\mathbb{C}$ is contained in $\{e_k(m) \mid k \in \mathbb{K}\}$.

By assumption (1), we also have $|\mathbb{C}| = |\mathbb{K}|$ and so $|\{e_k(m) \mid k \in \mathbb{K}\}| = |\mathbb{K}|$ follows.

But then the map

$$e_\cdot(m) \colon \mathbb{K} \to \mathbb{C}$$

is injective (also known as 1-1), from which (S2) follows.

Proof that perfect secrecy implies (S1)

Let n be $|\mathbb{K}|$. Since $|\mathbb{P}| = |\mathbb{K}|$, we may write $\mathbb{P}$ as a set $\{m_1, m_2, \ldots, m_n\}$ of n different plaintexts.

Fix a ciphertext c in $\mathbb{C}$. Then reorder the enumeration of $\mathbb{K} = \{k_1, k_2, \ldots, k_n\}$ (if needed) such that

$$\forall 1 \leq i \leq n: \qquad c = e_{k_i}(m_i)$$

This is possible since, by (S2) applied to c, for every m_i in $\mathbb{P}$ there is a unique k_i in $\mathbb{K}$ such that $c = e_{k_i}(m_i)$. Then we compute

$$\begin{aligned} p(P = m_i) &= p(P = m_i \mid C = c) && \text{by perfect secrecy} \\ &= \frac{p(C = c \mid P = m_i) \cdot p(P = m_i)}{p(C = c)} && \text{by Bayes' Theorem} \\ &= \frac{p(K = k_i) \cdot p(P = m_i)}{p(C = c)} && \text{as } k_i \text{ unique with } e_{k_i}(m_i) = c \end{aligned}$$

Since $p(C = c) > 0$, the above implies that $p(K = k_i) = p(C = c)$ for all $1 \leq i \leq n$. (QUESTION: does this argument also require $p(P = m_i) > 0$ and, if so, would that be a problem?)

But this shows that all $p(K = k_i)$ are equal to each other, and so have to equal $1/|\mathbb{K}|$ as this is a probability distribution. This shows (S1).

Example of perfectly secure cryptosystem: modified shift cipher

Alphabet is $\mathcal{A} = \{A, B, \ldots, Z\}$. Plaintexts and keys are finite words of fixed length $n \geq 1$ over that alphabet.

The probability distribution for keys is specified as follows: the key k equals $r_1 r_2 \ldots r_n$ where each r_i from $\mathcal{A}$ is chosen *independently and uniformly at random*.

Encryption of $m = a_1 a_2 \ldots a_n$ with such a key k is defined as $e_k(m) = c_1 c_2 \ldots c_n$ such that each c_i in $\mathcal{A}$ is the cyclic shift of a_i in $\mathcal{A}$ by $n(r_i)$, where $n(A)$ is 0, $n(B)$ is 1, and so forth.

For example, let n be 5, m be HELLO and k be FUIAT. Then the corresponding ciphertext is MYTLH. In the first position, e.g., the key is F which corresponds to 5 and so the shift of H is M.

It should be clear that $|\mathbb{P}| = |\mathbb{C}| = |\mathbb{K}| = 26^n$. By Shannon's Theorem, it suffices to show (S1) and (S2):

(S1) $p(K = k) = 26^{-n}$ since each r_i for k is chosen independently and uniformly at random.

(S2) It should be clear that for each m in $\mathbb{P}$ and each c in $\mathbb{C}$, there is a unique key k with $c = e_k(m)$.

Example of perfectly secure cryptosystem: one-time pad

Also known as Vernam cipher, patented by Gilbert Verman in 1917. Much later, it came to light that Frank Miller had already invented this cipher in 1882. For $n \geq 1$, set

$$\mathbb{P} = \{0,1\}^n = \mathbb{C} = \mathbb{K}$$

For a plaintext $m = m_1 m_2 \dots m_n$ and key $k = k_1 k_2 \dots k_n$, encryption is defined as

$$e_k(m) = m_1 \oplus k_1 \,||\, m_2 \oplus k_2 \,||\, \cdots \,||\, m_n \oplus k_n$$

where $||$ denotes concatenation of strings and $\oplus$ a function of type $\{0,1\}^2 \to \{0,1\}$, the *exclusive-or*:

$\oplus$	0	1
0	0	1
1	1	0

For example, $e_{011}(010) = 0 \oplus 0 \,||\, 1 \oplus 1 \,||\, 0 \oplus 1 = 001$.

(S2) We have $p(K = k) = 1/2^n$ since we assume that each k_i in a key k is chosen independently and uniformly at random.

(S1) Let $m = m_1 m_2 \dots m_n$ and $c = c_1 c_2 \dots c_n$. We have $e_k(m) = c$ for the key $k = k_1 k_2 \dots k_n$ with $k_i = m_i \oplus c_i$. EXERCISE: show that k is unique with that property.

Requirements for use of one-time pad

1. Key has to be *as long* as the message.
2. Key has to be *truly random*.
3. Key can be used *at most once*.

QUESTION: What if the last requirement is violated? Then Eve (a common name for an attacker) can launch a so called *chosen plain-text attack*:

i. Eve generates message m and asks Alice to encrypt it.

ii. Eve gets back ciphertext c, which Eve knows to be of form $m \oplus k$.

iii. Eve now recovers the key $k = c \oplus m$.

ASSUMPTION: Alice can be persuaded to do this encryption. And Alice will use k at least one more time.

The above attack generalises to values of n other than 1. Therefore, Eve can now decrypt any ciphertexts that were, or that will be, encrypted with that key k.

Chapters 13 of Nigel Smart's book: Block Ciphers

Now, plaintext m is divided into blocks of data m_i of equal length, typically 128-bits:

$$m = m_1 m_2 \dots m_k$$

The ciphers considered so far used 1-bit "blocks". Block ciphers assume cryptosystem $(e_k(\cdot), d_k(\cdot))$ for data of block length.

Key idea is to use $e_k(\cdot)$ to encrypt each data block m_i of m.

Modes of operation: Encryption of m_i may depend on encryption of m_j where $j < i$.

Example block cipher: DES "data encryption standard". From the mid 1970ies; 64-bit data blocks; 56-bit keys (too weak today), a US FIPS Standard

Recent block cipher: AES "advanced encryption standard". From 2000. US NIST Standard. 128-bit data blocks. Key size 128, 192 or 256 bits (security/performance tradeoff). Winner of a global, open competition (Belgian cryptographers won).

DES and AES are *iterated* block ciphers.

Iterations of Block Ciphers

Iterations of a block cipher are called "rounds".

For each round i, the encryption key k determines a "round key" k_i. The process of computing k_i from k is called the "key schedule".

Data blocks are encrypted by a "round function". Intuition:

$$\begin{aligned} \text{more rounds} &\Rightarrow \text{more security} \\ \text{longer keys} &\Rightarrow \text{more security} \end{aligned}$$

Example round function: *Feistel cipher*. Single data block $m = L_0 R_0$ split into a left half L_0 and right half R_0 of equal size. Encryption in round i is then:

$$L_i = R_{i-1} \qquad\qquad R_i = L_{i-1} \oplus F(K_i, R_{i-1})$$

In the above, F is the chosen Feistel function; $\oplus$ is exclusive-or applied bitwise in position; K_i is the key for round i.

DES is an instance of this, with its bespoke key scheduling and choice of Feistel function F.

Please study Figure 13.2 on page 244 of Nigel Smart's book.

Note: after the final round r, DES swaps L_r and R_r to prepare for its decryption mode. There is an exercise **(Exercise 8 in exercise sheet)** about that in the exercises sheet.

Security of Feistel Ciphers

1. Number of rounds: standards would proscribe these; again, there is here an issue of performance versus security.
2. Key schedule $KeySch(K, i) = K_i$: Is such a schedule secure? Is the schedule reversible for the decryption mode?
3. Feistel function F: needs to contain *non-linear* behavior to ensure enough information-theoretic resiliency.

Decryption for Feistel Cipher:

$$R_{i-1} = L_i \qquad\qquad L_{i-1} = R_i \oplus F(K_i, L_i)$$

In this equation i, R, and L refer to values during the encryption state. In particular, this means that decryption simply reverses the order of supplied round keys.

Decrypt using the *encryption* device of Figure 13.2 on page 244 of Nigel Smart's book:

i. Feed $R_r L_r$ as formal input $L_0 R_0$ into the encryption device.
ii. Reverse key schedule to $k_r, k_{r-1}, \ldots, k_1$ whilst running the encryption device on that input.

Advantage: Only have to implement or build an encryption device, no need for a decryption device, only need to manage D/E state.

Correctness of Feistel Ciphers

EXERCISE: **(Exercise 8 in exercise sheet)** show the correctness of this decryption for $r = 3$.

Let us now return to DES:

- $r = 16$ rounds
- 64-bit data blocks
- 56-bit keys with 48-bit round keys
- has its bespoke Feistel function

Study Figure 13.4 on page 247 of Nigel Smart's book.

Design of the DES Feistel function: Study Figure 13.5 on page 247 of Nigel Smart's book.

DES Key schedule: we refer to Nigel Smart's book for details on that; the point to remember is that this uses permutations and cyclic shifts based on the key and round number.

Security Analyses of Block Ciphers

DES and AES seem immune against:

- *differential cryptanalysis:* a *chosen-plaintext* attack that works by
  1. choosing m and m' with particular differences
  2. computing $c = e_k(m)$ and $c' = e_k(m')$
  3. analyzing the "difference" $c \oplus c'$
  4. repeating the first three steps as often as needed.
- *linear cryptanalysis:* approximate all non-linear components with linear functions; for example the S-boxes of DES; then do a probabilistic analysis about the key.

EXERCISE: **(Exercise 12 in exercise sheet)** Discuss the pluses and minuses of stream ciphers and block ciphers.

Recommended reading: read Chapter 13, Section 3 of Nigel Smart's book to get a feel for the AES design.

Modes of Operation

ECB mode "electronic code book":

$$\begin{aligned} m &= m_1 m_2 \ldots m_q \\ c &= c_1 c_2 \ldots c_q \qquad \text{where } c_i = e_k(m_i) \end{aligned}$$

Each block m_i is encrypted *independently* from any other block, but with the same key k.

Decryption: $d_k(\cdot)$ is used on each block c_i independently.

Real problem with this mode: If we encrypt m more than once with the same key, these ciphertexts are equal. Thus, this mode gives us *deterministic* encryption which attackers can exploit. Bad!

CBC Mode of Operation

CBC mode "cipherblock chaining". This requires an *initialization vector* IV of data block size. Encryption:

$$\begin{aligned} m &= m_1 m_2 \ldots m_q \\ c_1 &= e_k(m_1 \oplus IV) \\ c_i &= e_k(m_i \oplus c_{i-1}) \qquad \text{for all } 1 < i \leq q \end{aligned}$$

Note how the ciphertext c_i is a function of the key k, the plaintext block m_i, and the *previous ciphertext block* c_{i-1}. Decryption:

$$\begin{aligned} c &= c_1 c_2 \ldots c_q \\ m_1 &= d_k(c_1) \oplus IV \\ m_i &= d_k(c_i) \oplus c_{i-1} \qquad \text{for all } 1 < i \leq q \end{aligned}$$

So the ciphertext block c_i is decrypted by first decrypting it with key k and then forming the exclusive or of this result with the previous ciphertext block c_{i-1}. Note that IV plays the role of the "previous" ciphertext block for $i = 1$.

Both encryption and decryption for this mode require knowledge of the same IV (and of the same key k of course).

EXERCISE (in class): show the correctness of CBC decryption.

Mentimeter question: IV and secure use of block cipher.

OFB Mode of Operation: "output feedback"

This mode also requires use of an initialization vector IV. Although the length of a data block is fixed, that value is configurable:

$$m = m_1 m_2 \dots m_q \qquad \text{where each } m_i \text{ is } j \text{ bits long for fixed } j \geq 1$$

For $j = 1$ this is a so called "stream cipher". Encryption:

$$\begin{aligned} Y_0 &= IV \\ Y_i &= e_k(Y_{i-1}) \\ c_i &= m_i \oplus Y_i \end{aligned}$$

This uses the encryption device $e_k(\cdot)$ and initialization vector IV to generate a *key stream* $Y = Y_1 Y_2 \dots Y_q$. Encryption is then as for the one-time pad: $c = m \oplus Y$.

QUESTION: Why is the above not equal to the one-time pad? Why is the above not perfectly secure?

CTR Mode of Operation

CTR mode "counter". Uses initialization vector IV as a counter: $IV \simeq$ binary encoding of counter value. This adds 1 to IV for each subsequent encryption:

$$c_i = m_i \oplus e_k(IV + i)$$

Advantages:

1. This is a recent US NIST Standard.
2. We may compute all c_i in parallel, which is not possible in OFB, CFB, and CBC modes.
3. If $m_i = m_j$, then $c_i \neq c_j$ in general. So this improves over ECB.

Requirement: For l-bit data blocks, the plaintext message m must not be longer than 2^l bits.

QUESTION: How best to enforce this requirement in code?

Chapter 14 of Nigel Smart's book: Hash functions and MACs

$(e_k(\cdot), d_k(\cdot))$ gives us *confidentiality*.

What about *integrity* and *availability*? **(CIA)**

Integrity:

- plaintext may have been tampered with
- BIOS or OS kernel as "plaintext" at boot time, see e.g. the Trusted Platform Module (TPM) standard

Two (symmetric key) techniques:

1. Hash functions h: capture the integrity of m as a hash $h(m)$.
2. Message authentication codes (MACs): $MAC_k(m)$ as a *key-dependent* hash

Agency: Cannot say *who* created hash values. If this is an issue in an application, MACs are an alternative since a MAC is produced by someone who knows the used key.

Hash functions

$$h\colon \{0,1\}^* \to \{0,1\}^n \qquad\qquad h(x) = y$$

Input x is arbitrarily long. Output y is of *fixed length* n.

Cautionary note: Hash tables from programming languages and data structures are *different* from hash functions in cryptography. The latter have to satisfy important *security properties*:

Preimage resistance: Given y in the image of h, it should be computationally infeasible to compute an x with $h(x) = y$.

Intuition: finding such an x should require $\mathcal{O}(2^n)$ compute time.

Further security properties of hash functions

Collision resistance: It should be computationally infeasible to find two inputs x and x' (in particular, $x \neq x'$) such that $h(x) = h(x')$.

Intuition: Finding such a pair of colliding inputs should require $\mathcal{O}(2^{n/2})$ compute time. Why $n/2$ and not n here?

Answer: *birthday paradox*. The probability that 2 out of n people have the same birthday is

- 50% for $n = 23$
- 99.9% for $n = 70$
- 100% for $n \geq 367$

We can apply this paradox/principle to the sequence of hashes $h(x_1), h(x_2), \ldots$ for a stream of inputs $x_1, x_2, \ldots$ Note: we only need a collision of *any* two hashes in the former sequence, not a collision with the first hash in that sequence or another specific hash in that sequence.

Consequence of birthday paradox: For 80 bits of security in hash functions, we need at least $2 \cdot 80 = 160$ bits of output as $\mathcal{O}(2^{160/2}) = \mathcal{O}(2^{80})$.

Another security property: second preimage resistance

Given an input x, whose value is not controlled by an attacker, it should be computationally infeasible to find an input x' with $x \neq x'$ such that $h(x) = h(x')$.

A hash function h should possess all three security properties: preimage resistance, collision resistance, and second preimage resistance.

EXERCISE: **(Exercise 13 in exercise sheet)** Discuss what "computationally infeasible" might mean and whether this notion may evolve over time or depend on the nature of the attacker.

Mentimeter question: What is true (always or sometimes) about hash functions?

Understanding relative strengths of these security properties

We use "provable security" to analyze this: if one of these three security properties breaks for a hash function, does then another such property break as well for that hash function? Example:

Lemma (part of Lemma 14.2 on page 274 of Nigel Smart's book): Assume an oracle $\mathcal{O}$ such that for each y in the image of the hash function h, the oracle produces an random input x with $h(x) = y$. Then we can build a randomized algorithm that breaks *collision* resistance, makes use of the oracle $\mathcal{O}$, and is "computationally feasible".

PROOF: As usual, this assumes that calls to an oracle take constant time and space. The algorithm works as follows:

1. choose input x at random
2. compute $y = h(x)$
3. use oracle $\mathcal{O}$ to find x' with $h(x') = y$
4. if $x = x'$, go to 1. and repeat this process
5. output the pair (x, x') – a collision for h

Note: Step 1 above makes this a *randomized* algorithm. There typically are many x' with $h(x') = y$, certainly more than one. The algorithm terminates since the oracle is assumed to choose x' at random.

Exercises

1. **(Exercise 14.1 in exercise sheet)** Assume an oracle $\mathcal{O}$ that, for any y in the image of hash function h, produces a random x with $h(x) = y$. Assume further that calls to the oracle take constant time and space.

   Design a randomized algorithm that uses oracle $\mathcal{O}$, breaks second preimage resistance, and is efficient.

2. **(Exercise 14.2 in exercise sheet)** Show that a method that breaks second preimage resistance can be efficiently transformed into a method for breaking collision resistance.

Collision resistance does not imply preimage resistance

Let $g\colon \{0,1\}^* \to \{0,1\}^n$ be a collision-resistant hash function of n output bits. Then we define a hash function $h\colon \{0,1\}^* \to \{0,1\}^{n+1}$ of $n+1$ output bits from g as follows:

$$h(x) = \begin{cases} 0 \mid\mid x & \text{if } |x| = n \\ 1 \mid\mid g(x) & \text{otherwise} \end{cases}$$

We overload the symbol $|\cdot|$ to not just mean the size of a finite set, but also the length of a word/string (as in the definition above). Recall that $||$ denotes the concatenation of strings.

1. EXERCISE (in class): show that h is collision resistant since g is collision resistant.

2. h is *not* preimage resistant: this is true simply because we can easily invert all hashes that begin with bit 0 on the left.

   Note that preimage resistance means resistance against *all* possible outputs, so the property breaks if it breaks already one a sole output.

Hash function design: Merkle-Damgård construction

Given: a *compression function* $f\colon \{0,1\}^s \to \{0,1\}^n$ with $s > n$.

Assumption: compression function f *believed* to be collision resistant.

Construct hash function h out of f. Aim: h should be collision resistant.

EXERCISES: for these first study Algorithm 14.1 on page 276 of Nigel Smart's book.

(e1) **(Exercise 15.1 in exercise sheet)** Show that h in Algorithm 14.1. on page 276 is *not* collision resistant. Hint: let $s = 8$, $n = 4$, and consider the inputs $m_1 = 010$ and $m_2 = 0100$.

(e2) Do **(Exercise 15.2 in exercise sheet)**.

Design of compression function f

Avalanche effect desired: small changes in the input should result in unpredictable changes in the output.

The MD4 algorithm is basis for design of several such functions.

Current state of affairs for hash functions:

- MD5 and SHA-0 are broken.
- SHA-2 has theoretical attacks only.
- US NIST pushed for a new standard SHA-3, released in August 2015. Need for such a standard (now and that quickly) was questioned.

  The argument for a new standard seems to be that SHA-3 is now ready to use should SHA-2 be really broken.
- In February 2017, a theoretical attack on SHA-1 was leveraged to find an SHA-1 collision! Involved 6,610 years of processor time!!

Building hash functions from block ciphers

For example, the Davies-Meyer hash: given plaintext $m = x_1 x_2 \ldots x_t$, where last block x_t is padded to block size if needed.

Given furthermore a block cipher $E_K(\cdot)$ with key K and l-bit data blocks, i.e. each x_i above is a block of l bits. Then we define

$$\begin{aligned} H_0 &= IV \\ H_i &= f(x_i, H_{i-1}) \\ h(m) &= H_t \end{aligned}$$

Note: the initialization vector IV is now *public* and *constant*: it has the same value each time the hash function is used, and this value is known to all.

This is in contrast to use of the IV in encryption, where the IV should only be known to those who know key K, and the IV should not be reused for further encryptions.

We build compression function f using block cipher $E_K(\cdot)$ and data blocks as keys:

$$f(x, H) = E_x(H) \oplus H$$

Note that the keys are not random, but the $\oplus$ is hoped to hide such non-random information.

From block cipher to hash function: Matyas-Meyer-Oseas hash

Plaintext $m = x_1 x_2 \dots x_t$, block cipher $E_K(\cdot)$, and computation of H_i and $h(m) = x_t$ is the same as for the Davies-Meyer hash.

The difference is in the definition of the compression function f:

$$f(x, H) = E_{g(H)}(x) \oplus x$$

Now:

1. x is used as "data" and not as key
2. $g\colon \{0,1\}^n \to \mathbb{K}$ is a *key derivation function* (KDF)

The design of KDFs is an entire topics of its own, not covered here.

Message Authentication Codes (MAC)

Alice (A) sends Bob (B) a message m and its *integrity tag* $h(m)$:

$$A \longrightarrow B\colon m,\ h(m)$$

where h is a hash function.

- Assumption: Bob knows that h is used to create such tags.
- Bob receives message m' and integrity tag t.
- Bob computes $h(m')$. If $t \neq h(m')$, this violates integrity of received message.
- Otherwise ($t = h(m')$), Bob can be sure that Alice sent him m', right?

No! Eve could intercept the message $m, h(m)$, create her own message $m^*, h(m^*)$, and insert her message into the network as the one to be sent to Bob. Bob, following the above protocol, would accept integrity of m^*!

Problem Analysis: we need data integrity *with agency* in these situations.

Solution: make the hash *key-dependent*, and let Alice and Bob share that key.

MAC idea

$$code = MAC_K(m)$$

k is secret, m is not secret

MAC codes give us integrity, not confidentiality. An idea for both confidentiality and integrity:

$$e_{k_1}(m) \;||\; MAC_{k_2}(e_{k_1}(m))$$

Note:

1. key for confidentiality (k_1) $\neq$ key for integrity (k_2).

   QUESTION: why is this important?

2. the above gives us integrity of ciphertext, not of plaintext; a potential problem as this "signs what is said, not what is meant".

   Please read up on *Horton's Principle* on Wiki.

Security Requirements on MACs

Only those agents who know the key k can:

1. produce MACs of form $MAC_k(m)$
2. verify integrity of MACs of form $MAC_k(m)$

In particular, an attacker in possession of $MAC_k(m)$ should be unable to produce a message $m' \neq m$ with $MAC_k(m) = MAC_k(m')$.

Note: MACs have keys so that we get *agency* of integrity. But keys require *key management*.

Doing key management securely is a hard problem on real systems.

Using hash function to build a MAC

Given: hash function h, collision free. Building a MAC from h requires care. For example, let h be iterated (e.g. Merkle-Damgård):

$$H_{i+1} = f(H_i \,||\, m_i)$$

with compression function f.

Above is subject to *length-extension attack*: let m_1 be message with t blocks, and m_2 a sole block (chosen by the attacker).

For $c_1 = MAC_k(m_1)$, which equals $h(k \,||\, m_1)$, the attacker can now compute MAC of $m_1 \,||\, m_2$, violating MAC security:

$$\begin{aligned}
c_2 &= MAC_k(m_1 \,||\, m_2) \\
&= h(k \,||\, (m_1 \,||\, m_2)) \\
&= h((k \,||\, m_1) \,||\, m_2) \\
&= f(h(k \,||\, m_1) \,||\, m_2) \qquad \text{as } m_2 \text{ sole block and } H_{i+1} = f(H_i \,||\, m_i) \\
&= f(c_1 \,||\, m_2) \qquad \text{but attacker knows } f,\ c_1, \text{ and } m_2!
\end{aligned}$$

More on attacking MACs based on iterated hash functions

EXERCISES:

1. The last attack could seemingly be prevented if the message m_1 is concatenated with a sole block b so that m_1, b is communicated, where b expresses the bit-length of m_1 in binary. The MAC of m_1 would still be $c_1 = MAC_k(m_1)$. SHOW: an attacker who knows c_1 can still compute a new MAC of form $MAC_k(m_1 \,||\, b \,||\, m_2)$ and construct the corresponding message for communication.

2. **(Exercise 16 in exercise sheet)** Show that a definition $MAC_k(m) = h(m \,||\, k)$, which *appends* the key to the message before hashing, is also not secure.

Good proposal for hash function design, and a standard is: HMAC, where p_1 and p_2 are used to pad inputs to block size:

$$HMAC_k(m) = h(k \,||\, p_1 \,||\, h(k \,||\, p_2 \,||\, m))$$

Using block ciphers to build MACs

Example: message $m = m_1 m_2 \ldots m_q$ of q data blocks of n bits each.

CBC-MAC used a block cipher $e_k(\cdot)$ in CBC mode, where m_q is padded to block size if needed:

$$\begin{array}{ll} I_1 = m_1 & O_1 = e_k(I_1) \\ I_i = m_i \oplus O_{i-1} & O_i = e_k(I_i) \\ \text{CBC-MAC}_k(m) = \mathit{postproc}(O_q) & \end{array}$$

postproc may truncate output to $m < n$ bits and perform other postprocessing, for example

$$\mathit{postproc}(O_q) = e_k(d_{k_1}(O_q)) \qquad \text{or} \qquad \mathit{postproc}(O_q) = e_{k_1}(O_q)$$

with *new* key k_1 used just in this postprocessing.

Padding in m_q requires care, discussed next.

CBC-MAC and padding

- Method 1: fill up rest of last block with 0 bits.
- Method 3: as in Method 1, but this also adds a block with binary encoding of bit-length of *unpadded* message.

EXERCISES:

1. **(Exercise 17 in exercise sheet)** Compare both padding methods above in terms of their pluses and minuses.
2. **(Exercise 18 in exercise sheet)** Let $m = m_1 m_2 \dots m_q$ and $M = \text{CBC-MAC}_k(m_1 m_2 \dots m_q)$ where m_q is padded as in Method 1 above. Show that M also equals $\text{CBC-MAC}_k(m \mid\mid M \oplus m_1 \mid\mid m_2 m_3 \dots m_q)$, and explain why this makes this MAC insecure. [Spoiler alert: the solution is on the next slide, but try this first yourself.]

Solving last exercise on previous slide

We know that $M = O_q$ since this processed the first q blocks of $m = m_1 m_2 \ldots m_q$.

By definition of CBC-MAC, we then continue to compute

$$\begin{array}{lcll} I_{q+1} & = & (M \oplus m_1) \oplus O_q & \text{exclusive or of } (q+1)\text{th block and previous } O \\ & = & (M \oplus m_1) \oplus M & \text{by above} \\ & = & m_1 & \text{property of } \oplus \\ & = & I_1 & \text{by def. of CBC-MAC} \end{array}$$

The CBC-MAC computation therefore renders I_1 for input $m \,||\, M \oplus m_1$, and this computation still has to process $m_2 m_3 \ldots m_q$.

Therefore, this remaining computation starts in the same state as the one for computing M after m_1 has been processed.

This implies that $O_{2q} = Q_q = M$ as claimed.

Conclusion: post-processing adds genuine security.

Exercise (Exercise 19 in exercise sheet)

Even padding, as seen in Method 3, can result in length-extension attacks.

Assumption: attacker has MAC M_1 of message m_1, and MAC M_2 of message m_2, and both messages m_1 and m_2 are sole blocks.

Notation: $P(n) =$ data block that encodes natural number n in binary.

Show the following:

1. M_1 equals $e_k(e_k(m_1) \oplus P(b))$ where b is block size of block cipher $e_k(\cdot)$.
2. Let m_3 be a new sole data block. Then CBC-MAC$_k(m_1 \,||\, P(b) \,||\, m_3)$ equals

$$e_k(e_k(e_k(e_k(m_1) \oplus P(b)) \oplus m_3) \oplus P(3b))$$

3. Let m be the 3-block message $m_2 \,||\, P(b) \,||\, m_3 \oplus M_1 \oplus M_2$. Then

$$\text{CBC-MAC}_k(m) = \text{CBC-MAC}_k(m_1 \,||\, P(b) \,||\, m_3)$$

Chapters 15+16 of Nigel Smart's book: public-key cryptography

Motivation:

- symmetric key needs to be communicated securely to any decrypting party – over an insecure channel.
- public-key crypto solves this key management problem, and introduces another problem – that of *key authentication*

Idea: Agent A has a pair of keys $(pub_A, priv_A)$ where

- pub_A is agent A's *public* key: all can know this key in principle
- $priv_A$ is agent A's *private* or *secret* key: only A should know this key

Naive (insecure) protocol for how agent B can share a symmetric key K with agent A:

1. $B \longrightarrow A$: $\quad pub_A(K)$
2. A uses $priv_A$ to compute K from step 1.

Requirements:

(1) $priv_A(pub_A(K)) = K$

(2) $priv_A$ remains secret even if pub_A and $pub_A(K)$ are public

RSA

First and important public key crypto system: see textbook for its US/UK history.

Its key generation works as follows:

1. compute large primes p and q at least ≥ 1024 bits
2. compute the so called *RSA modulus* $N = p \cdot q$
3. choose e with $gcd(e, (p-1) \cdot (q-1)) = 1$
4. compute d with $d \cdot e = 1 \mod (p-1) \cdot (q-1)$, then

$$pub_A = (e, N) \qquad priv_A = (d, N)$$

Notes:

- in $priv_A$ only d is secret
- d is computed used *Extended Euclid* algorithm where item 3 above ensures existence of d
- convert d into a positive equivalent if output of that algorithm is negative
- choice of e may impact security of RSA

RSA cipher

$$\begin{aligned}\mathbb{P} &= \{0, 1, \ldots, N-1\} = \mathbb{C} \\ \mathbb{K} &= \{(e, N), (d, N)\} \qquad \text{but where } (d, N) \text{ is secret}\end{aligned}$$

- Encryption: $e_{(e,N)}(m) = m^e \mod N$
- Decryption: $d_{(d,N)}(c) = c^d \mod N$

Correctness of cipher: By Lagrange's Theorem, $x^{\phi(N)} = 1 \mod N$ where

$$\phi(N) = |\{x \mid 1 \leq x \leq N, gcd(x, N) = 1\}|$$

is the *Euler totient function*.

Note: $\phi(N) = \phi(p \cdot q) = (p-1) \cdot (q-1)$. Also, from $d \cdot e = 1 \mod (p-1) \cdot (q-1)$ we get that there is some s with $ed = 1 + s \cdot (p-1) \cdot (q-1)$. Now, we may compute:

$$\begin{aligned}d_{(d,N)}(e_{(e,N)}(m)) &= (m^e)^d \mod N \\ &= m^{e \cdot d} = m^{1+s \cdot (p-1) \cdot (q-1)} \\ &= m \cdot (m^{(p-1) \cdot (q-1)})^s = m \mod N\end{aligned}$$

RSA worked example

Let $p = 7$ and $q = 11$ (definitely not more than 1024 bits, values of p and q are unrealistically small!). Then $N = 77$ and

$$\mathbb{P} = \mathbb{C} = \{0, 1, \ldots, 76\}$$

Choose $e = 37$. Use Extended Euclid to compute $d = 13$. Note that $37 \cdot 13 = 481 = 1 \mod 60$.

Let $m = 2$. Then:

(1) $c = m^e = 2^{37} = 51 \mod 77$

(2) $c^d = 51^{13} = 2 = m \mod 77$ as claimed

Security of RSA

FACTORING problem: given N, known to be the product of two primes, find these primes:

given an RSA modulus N, find primes p and q with $N = p \cdot q$

RSA problem:

- given $N = p \cdot q$ with primes p and q
- given e with $gcd(e, (p-1) \cdot (q-1)) = 1$
- given c in $\{0, 1, \ldots, N-1\}$
- *find:* m in $\{0, 1, \ldots, N-1\}$ such that $m^e = c \mod N$

Lemma 11.4: The RSA problem is no harder than the FACTORING problem.

PROOF **(Exercise 24 in exercise sheet)**: Show that we can solve the RSA problem once we can solve the FACTORING problem (i.e. once we know p and q).

A real problem of RSA

$m_1 = m_2$ implies that $m_1^e = m_2^e \mod N$ and so RSA encryption is *deterministic*.

This is similar to the problem of deterministic encryption for block ciphers in ECB mode.

Standard solution:

- "blind" the input m with random number r
- this requires care and use of other primitives such as hash functions and pseudo-number generators
- see Chapter 12.6 in the book *Cryptography Engineering* by Ferguson, Schneier, and Kohno (Wiley, March 2010) for an example of this.

Key exchange

Alice and Bob want to share key K for symmetric encryption. Assumptions:

1. Alice and Bob do not want to rely on a trusted third party.
2. Alice and Bob don't have or don't want to use *longterm keys* for this exchange of a *session key* K.

Solution: Diffie-Hellman Key Exchange protocol:

- given: prime p of at least 1024 bits
- given: g in $\{2, 3, \ldots, p-1\}$ such that $\{g^a \mod p \mid 0 \leq a \leq p-1\}$ is "large"

(1) Alice generates random a, sends $g^a \mod p$ to Bob.

(2) Bob generates random b, sends $g^b \mod p$ to Alice.

(3) Bob computes $K_1 = (g^a)^b \mod p$ as shared key. Alice computes $K_2 = (g^b)^a \mod p$ as shared key.

- By algebra, we have $K_1 = K_2$, so they really share the same key.

Diffie-Hellman Key exchange example

$p = 2,147,483,659$ this is unrealistically small!
$g = 2$

QUESTION: why do we need assurance that $\{2^a \mod p \mid 0 \leq a \leq p-1\}$ is large?

(1) Alice generates $a = 12,345$ at random and sends $A = g^a \mod p = 428,647,416$ to Bob.

(2) Bob generates $b = 654,323$ at random and sends $B = g^b \mod p = 450,904,856$ to Alice.

(3) Bob computes $A^b = 428,647,416^{654,323} \mod p = 1,333,327,162$. Similarly, Alice computes $B^a = 450,904,856^{12,345} \mod p = 1,333,327,162$.

Therefore, the shared key is $K = 1,333,327,162$.

Security of this protocol: Depends on the ...

Discrete Logarithm Problem:

"For appropriate choices of prime p and element g, it is hard to compute a knowing $g^a \mod p$."

Discussion of Diffie-Hellman Key exchange

- works in any mathematical group in which operation $f(a) = g^a$ is hard to "invert"
- both agents generate parts of the shared key, making this more trustworthy
- satisfies *forward secrecy:* if any longterm keys of Alice or Bob get compromised, it won't compromise secrecy of shared key K
- may use p with about 160 bits for certain groups based on *elliptic curves*; this is a *performance gain* used, e.g. in cryptocurrency **Bitcoin**
- often need to trim a key – such as $(g^a)^b \mod p$ – down to needed bit size (e.g. 128 bits): hash functions are doing this job well
- Diffie-Hellman protocol was in the paper that proposed idea of public-key cryptography in 1976 – an idea apparently already known within UK GCHQ at that time

Diffie-Hellman Key exchange: man in the middle attack

Assumptions:

- attacker Eve convinces Alice that Eve is actually Bob.
- attacker Eve also convinces Bob that Eve is actually Alice

Then Eve runs and *interleaves* two different runs of this protocol. The first run is initiated by Alice, the second one by Eve – pretending to be Alice:

Alice		*Eve*		*Bob*
a	$\longrightarrow$	g^a		
g^m	$\longleftarrow$	m		
g^{am}		g^{am}		
		n	$\longrightarrow$	g^n
		g^b	$\longleftarrow$	b
		g^{bn}		g^{bn}

Now, Alice shares key g^{am} with Eve (but Alice thinks she shares this key with Bob!). And Bob shares key g^{bn} with Eve (where Bob thinks he shares this key with Alice!).

Digital signatures

Can avoid man in the middle attack if Alice and Bob also sign their message before sending. May use RSA to do this:

- *signature generation:* message + private key $\rightsquigarrow$ signature
- *signature verification:* message + signature + public key $\rightsquigarrow$ "accept" or "reject"

Use of RSA for this is possible since it satisfies $pub_A(priv_A(m)) = m$ as well.

- Alice's signature generation: $s = priv_A(m) = m^d \mod N$
- verification of Alice's claimed signature s: does m equal $s^e \mod N$?

Problems:

- message m may be too long to fit into $\mathbb{P}$ of RSA (sign hash instead)
- how do we know it is m and not some other m' that we verify?

RSA signature with hash

Recall that Alice's RSA keys are $pub_A = (e, N)$, $priv_A = (d, N)$, and let h be a hash function.

Alice's signature of m is the pair (m, s) where

$$s = h(m)^d \mod N$$

Signature verification:

- given a pair (m, s) as claimed signature

1. compute $h' = s^e \mod N$
2. compute $h(m)$ by extracting m from pair (m, s)
3. accept signature if $h(m)$ equals h', and reject it otherwise

Implementation issue: $h(m)$ may not be distributed well over $\mathbb{P}$; see Chapter 12.7 in book by Ferguson, Schneier, and Kohno for one way of addressing this.

RSA signature with hash: **(Exercise 28 in exercise sheet)**

Assume an RSA signature scheme based on a hash function – as just described.

Analyze what capabilities an attacker might have when the following properties of hash function h are *not* true:

(1) preimage resistance

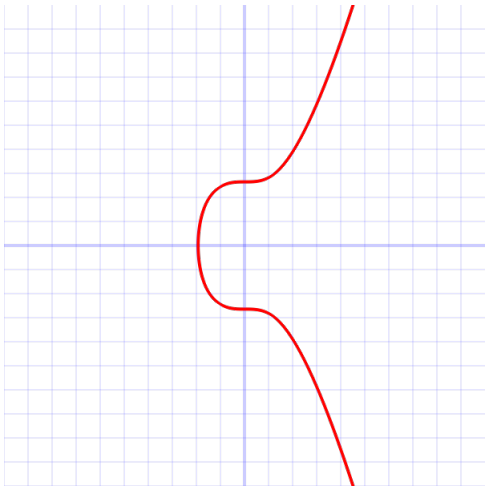
(2) collision resistance

(3) second preimage resistance

Motivating next topic: Elliptic Curve `Secp256k1` used in Bitcoin

- defined in "Standards for Efficient Cryptography SEC 2: Recommended Elliptic Curve Domain Parameters" by Certicom Research
- used for *Digital Signature Algorithm* ECDSA
- chosen over US NIST standards because:
  - allows for more efficient computations
  - constants determined in predictable way, making "back doors" less likely
- preferred to RSA since security of ECDSA believed to require less bits in chosen prime numbers

Elliptic Curves for Public Key Cryptography



- prime p is $2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$
- elliptic curve above[2] defines points (x, y) of *integer* coordinates such that

$$y^2 \mod p = (x^3 + 7) \mod p$$

- addition on curve: the *group law* $P_1 + P_2$ for points P_1 and P_2 on the curve – ***mathematical details covered later on***
- multiplication on curve: for $k > 1$, set $k \star P$ as addition of k many copies of P:

$$k \star P = P + \cdots + P$$

- note: $k \star P$ is sometimes written as $[k]P$, e.g. in Nigel Smart's book
- intuition: given points P and Q on curve, it is hard to determine whether there is some k with $Q = k \star P$, and to compute such k
- Similar to *Discrete Logarithm problem*: given g and h, compute k such that $g^k \mod q = h$ for suitable g and prime q

[2]Image source: `https://en.bitcoin.it/w/images/en/b/bf/Secp256k1.png`

Application example: From private key to Bitcoin address

- G generator of prime order $q > 2^{160}$, k random 256 bit number as private key, $K = k \star G$ public key
- bitcoin address for private key k is

$$A = RIPEMD{-}160(SHA{-}256(K))$$

- first harden input for RACE Integrity Primitives Evaluation Message Digest (RIPEMD) with SHA-256, this is *risk management*:
- composition means flaw in either SHA-256 or RIPEMD-160 not likely to create flaw in ECDSA that uses such hashes
- next: understand how messages are signed, and how signatures are verified

ECDSA Signature Generation

- G generator of prime order $q > 2^{160}$, k random 256 bit number as private key, $K = k \star G$ public key
- m message to be signed
- need to map points P on elliptic curve to integer interval $[0, q-1]$:

$$f(P) = \text{“}x\text{-coordinate of point } P\text{”} \mod q$$

- generate signature as follows:

1. $h = RIPEMD{-}160(SHA{-}256(m))$
2. choose random u, an *ephemeral* key, with $0 < u < q$
3. $r = f(u \star G)$ (repeat step 2 if $r = 0$)
4. $s = (h + k \cdot r) \cdot u^{-1} \mod q$ (repeat step 2 if $s = 0$)

Signature of m is pair (r, s).

QUESTIONS:

- are these digital signatures deterministic?
- why is bit length of order of generator at least as large as number of bits (160) of message hash?

ECDSA Signature Verification

- G generator of prime order $q > 2^{160}$, k random 256 bit number as private key, $K = k \star G$ public key
- is pair (r, s) valid signature of message m?
- use following protocol to verify this:

1. $h = RIPEMD{-}160(SHA{-}256(m))$
2. $a = h \cdot s^{-1} \mod q$
3. $b = r \cdot s^{-1} \mod q$
4. $v = f(a \star G + b \star K)$
5. accept if $v = r$

Recall that $f(P) =$ "x-coordinate of point P" mod q for points P on elliptic curve

Note that addition in $a \star G + b \star K$ is that of the elliptic curve.

EXERCISES:

- ECDSA is correct: valid signatures do get verified successfully.
- Using the same ephemeral key u for signing two different messages m and m' allows an attacker to compute the private key k.

ECDSA Security

Security of this digital signature scheme depends on hardness of the analogue of the Discrete Logarithm Problem for elliptic curves:

Elliptic Curve Discrete Logarithm Problem: Given two points P and Q on an elliptic curve such that Q is a multiple of P, it is a hard problem to find some k such that $Q = k \star P$.

Some of the algorithms that attempt to compute discrete logarithms for the groups based on numbers modulo a prime p can be adjusted to elliptic curves.

This is one reason why choice of parameters of an elliptic curve is important.

Trust issues:

- who generated such parameters?
- can the integrity of such parameters be verified?
- do parameter choices give good trade-offs between level of security and computational efficiency?

Chapter 4 of Nigel Smart's book: Elliptic Curves

Will now look at elliptic curves and their mathematics in some more detail.

Requires to work with *fields*: a field F is algebraic structure $(F, +, *, 0, 1)$ where

- $(F, +, 0)$ is an abelian group
- $(F \setminus \{0\}, *, 1)$ is an abelian group
- $*$ distributes over $+$

Examples:

- the rational numbers $\mathbb{Q}$ with the usual meaning of 0, 1, $+$, and $*$ form a countably infinite field
- the real numbers $\mathbb{R}$ with the usual meaning of 0, 1, $+$, and $*$ form an uncountable field
- the complex numbers $\mathbb{C}$ with the usual meaning of 0, 1, $+$, and $*$ form a field; unlike the first two, it has no linear order
- let p be a prime number, then the set of numbers $\mathbb{F}_p = \{0, 1, \ldots, p-1\}$ where $+$ and $*$ are understood to be modulo p form a finite field

Finite fields

Concepts in finite fields: let $(K, +, *, 0, 1)$ be a finite field.

- the *order* of K is the size of set K
- the *characteristic* of K is the smallest $n > 1$ such that the $n$fold sum $1 + \cdots + 1$ equals 0

QUESTION: why is the characteristic well defined for finite fields, and always greater than 1?

Facts about finite fields K:

- there is some prime p and $n \geq 1$ such that the order of K is p^n
- two finite fields with the same order are isomorphic as fields
- fields of characteristic 2 are exactly those of order 2^n for some $n \geq 1$

Projective plane

Let $(K, +, \cdot, 0, 1)$ be a finite field. Its *projective plane* is the set

$$\mathbb{P}^2(K) = \{(X, Y, Z) \in K^3 \mid \text{at least one of } X,\ Y,\ Z \text{ is non-zero}\} \quad (3)$$

Define equivalence relation $\equiv$ on $\mathbb{P}^2(K)$:

$$(X, Y, Z) \equiv (X', Y', Z') \text{ iff } \exists \lambda \in K \colon X = \lambda \cdot X', Y = \lambda \cdot Y', Z = \lambda Z' \quad (4)$$

So two points on the projective plane are equivalent if we can convert one into the other by a "scalar multiplication".

EXERCISE: show that $\equiv$ is indeed an equivalence relation on $\mathbb{P}^2(K)$.

Example: let $K = \mathbb{F}_7$, the numbers modulo 7 where $+$ and $\cdot$ are modulo 7 as well. Then

$$(4, 1, 1) \equiv (5, 3, 3)$$

QUESTION: what is the λ in $\mathbb{F}_7$ that realizes the above equivalence?

Homogeneous Weierstrass equations

These are a form of elliptic curve

$$E\ :\ Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \qquad (5)$$

where a_1, a_2, a_3, a_4, a_6 are in K.

Note: E is a continuous curve as familiar from calculus, when K is $\mathbb{R}$.

We are interested in points on E that are from the projective plane:

$$E(K) = \{(X,Y,Z) \in \mathbb{P}^2(K) \mid Y^2Z+a_1XYZ+a_3YZ^2 = X^3+a_2X^2Z+a_4XZ^2+a_6Z^3 \text{ holds } \} \qquad (6)$$

Note:

- both E and $E(K)$ may be called "the elliptic curve", the latter somewhat inappropriately
- a point (X,Y,Z) on $E(K)$ is often identified with its equivalence class for $\equiv$, or with other points that it is equivalent with

EXERCISES:

- Let $(X,Y,Z) \equiv (X',Y',Z')$. Then (X,Y,Z) is in $E(K)$ iff (X',Y',Z') is in $E(K)$.
- Let (X,Y,Z) be in $E(K)$ and $Z = 0$. Then $(X,Y,Z) \equiv (0,1,0)$ where 0 and 1 are the respective elements of K of course.

Point $\mathcal{O} = (0,1,0)$ is *"point at infinity"* of $E(K)$.

Affine version of Weierstrass equation

Equation "(5)" in Nigel Smart's book, which treats Z as 1 in (5) of our previous slide:

$$(5Smart) \;\; E : \; Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6$$

Solutions to E in affine case are all solutions to above equation, and $\mathcal{O}$:

$$\{\mathcal{O}\} \cup \{(X, Y) \in K^2 \mid Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6 \text{ holds } \}$$

Engineering for efficiency:

Affine form is basis for many cryptographic protocols, for example its use in Bitcoin.

But switch from K^2 to $\mathbb{P}^2(K)$ improves computational efficiency, as we will see.

Mappings between projective plane $\mathbb{P}^2(K)$ and K^2

One type of mapping is $f\colon K^2 \mapsto \mathbb{P}^2(K)$ and $g\colon \mathbb{P}^2(K) \mapsto K^2$, where

$$\begin{aligned} f(\mathcal{O}) &= \mathcal{O} \\ f((X,Y)) &= (X\cdot Z, Y\cdot Z, Z) \qquad \text{for } (X,Y)\neq\mathcal{O} \text{ and } Z\neq 0_K \text{ random} \\ g(\mathcal{O}) &= \mathcal{O} \\ g((X,Y,Z)) &= (X/Z, Y/Z) \qquad \text{where } Z\neq 0_k \text{ and division is that in } K \end{aligned}$$

Definitions in K^2 are easier to understand, computations in $\mathbb{P}^2(K)$ often more efficient: *less multiplicative-inverse computations*.

Another type of mapping transforms (X,Y,Z) in $\mathbb{P}^2(K)$ with $Z\neq 0_K$ to

$$(X/Z^2, Y/Z^3)$$

which is sometimes better computationally.

Simpler elliptic curve form for characteristic $\neq 2, 3$

Our assumption in using elliptic curves for cryptography here is that the characteristic of K is different from 2 and from 3.

In particular, the order of K is not a power of 2. And the assumption is met for orders p where p is a large prime (as in the Elliptic Curve `Secp256k1` used in Bitcoin).

This assumption allows us to change representation of variables X and Y as follows:

$$X = X' - \frac{b_2}{12} \tag{7}$$

$$Y = Y' - \frac{a_1}{2} \cdot (X' - \frac{b_2}{12}) - \frac{a_3}{2} \tag{8}$$

where $b_2 = a_1^2 + 4a_2$.

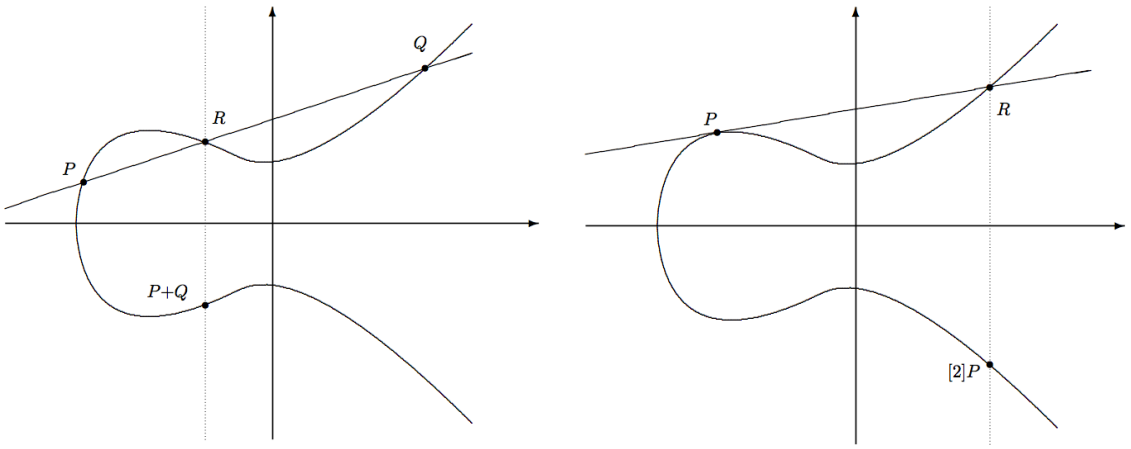
QUESTION: where in (7) do we use the assumption that the characteristic is neither 2 nor 3?

Transformation (7) turns (5Smart) into *isomorphic* elliptic curve

$$E : Y^2 = X^3 + aX + b \qquad a, b \in K \text{ constants} \tag{9}$$

Can define the important *group law* on this short form next.

Group law of elliptic curve: the chord-tangent process



(Figures 4.1 and 4.2 from Nigel Smart's book.)

1. For $P \neq \mathcal{Q}$ on curve, line through P and Q intersects curve at exactly one other point R. Reflect that point at x-axis to get $P + Q$.

2. For $P = Q$ we have two cases:

   (a) Tangent of P on curve intersects curve at another point (there will be at most one such point R). Reflect R on x-axis to get $P + P$.

   (b) Tangent of P on curve does not intersect another point on curve. We say it "intersects at infinity" and define $P + P = \mathcal{O}$.

For $P + P + \cdots + P$ with n summands, we write $n \star P$. Note that Smart's book (and the right figure above) write $[n]P$ instead.

Algebraic description of Group Law

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ from K^2 be points on the curve $E(K)$ defined in (9), recall that

$$E : \ Y^2 = X^3 + aX + b \qquad\qquad a, b \in K \text{ constants}$$

Then we have, whenever $P_1 + P_2 \neq \mathcal{O}$, that

$$P_1 + P_2 = (\lambda^2 - x_1 - x_2, -\lambda \cdot x_3 - \mu) \tag{10}$$

where λ and μ are defined by cases and x_3 equals $\lambda^2 - x_1 - x_2$. First, we express the reflection along the x-axis by $-P_1 = (x_1, -y_1)$. The cases are then:

- Let $x_1 = x_2$ and $P_2 \neq -P_1$. Then

$$\lambda = \frac{3x_1^2 + a}{2y_1} \qquad\qquad \mu = \frac{-x_1^3 + ax_1 + 2b}{2y_1}$$

- Let $x_1 \neq x_2$. Then

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

Exercise

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on a curve of form (9) such that $P_1 + P_2 \neq \mathcal{O}$.

1. Let $(x_3, y_3) = P_1 + P_2$.

   (a) Give explicit formulas for x_3 and y_3 in terms of $\{x_1, x_2, y_1, y_2\}$ when $x_1 \neq x_2$.

   (b) Give explicit formulas for x_3 and y_3 in terms of $\{x_1, x_2, y_1, y_2\}$ when $x_1 = x_2$ and $P_2 \neq P_1$.

2. Show that the algebraic definition of the Group Law computes a point $P_1 + P_2$ that is on the curve.

Example of curve in short Weierstrass form

Let finite field K be $\mathbb{F}_7$. Consider the curve

$$E : Y^2 = X^3 + X + 3 \qquad \text{where } a = 1 \text{ and } b = 3$$

Up to equivalence $\equiv$, this curve has six points:

$$\mathcal{O}, (4,1), (6,6), (5,0), (6,1), (4,6)$$

Group law determines abelian group (from page 72 of Nigel Smart's book):

$+$	$\mathcal{O}$	$(4,1)$	$(6,6)$	$(5,0)$	$(6,1)$	$(4,6)$
$\mathcal{O}$	$\mathcal{O}$	$(4,1)$	$(6,6)$	$(5,0)$	$(6,1)$	$(4,6)$
$(4,1)$	$(4,1)$	$(6,6)$	$(5,0)$	$(6,1)$	$(4,6)$	$\mathcal{O}$
$(6,6)$	$(6,6)$	$(5,0)$	$(6,1)$	$(4,6)$	$\mathcal{O}$	$(4,1)$
$(5,0)$	$(5,0)$	$(6,1)$	$(4,6)$	$\mathcal{O}$	$(4,1)$	$(6,6)$
$(6,1)$	$(6,1)$	$(4,6)$	$\mathcal{O}$	$(4,1)$	$(6,6)$	$(5,0)$
$(4,6)$	$(4,6)$	$\mathcal{O}$	$(4,1)$	$(6,6)$	$(5,0)$	$(6,1)$

This group is isomorphic to $(\mathbb{Z}_6, +, 0)$ here.

Exercise

1. The elliptic curve `Secp256k1` used in Bitcoin is

$$E : Y^2 = X^3 + 7 \qquad \text{where } a = 0 \text{ and } b = 7 \text{ in (9)}$$

   Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on that curve such that $P_1 + P_2 \neq \mathcal{O}$. Therefore, we may write

$$P_1 + P_2 = (x_3, y_3)$$

   Give explicit formulas for x_3 and y_3 in terms of the coordinates from $\{x_1, x_2, y_1, y_2\}$, making case distinctions as required.

2. **(Exercise 15.1-4 in exercise sheet)** Let the Elliptic Curve be $E : Y^2 = X^3 + X + 3$ and let the finite field K be $\mathbb{F}_7$.

   (a) Show that $(4, 1)$ is in $E(\mathbb{F}_7)$.

   (b) Show that $(4, 6)$ is in $E(\mathbb{F}_7)$.

   (c) Appeal to algebra to explain why $(4, 1) + (4, 6)$ equals $\mathcal{O}$ in E.

   (d) Appeal to the algebraic definition of $+$ on E to compute in detail the result of $(4, 1) + (6, 6)$, which is $(5, 0)$.

Elliptic curve parameter choices for cryptography

Let K be a finite field of order p^n for prime p. Recall $E(K)$ is the set of points from K^2 on a curve E of form (9), and so $|E(K)|$ is the size of that set.

Trace of Frobenius at p^n is defined as

$$t = p^n + 1 - |E(K)| \tag{11}$$

1. Let $n = 1$ and so p^n is prime. Let the trace of Frobenius for p be 1. Then the curves E are cryptographically weak, don't use them!

2. There are similar parameter combinations that should be avoided when $n > 1$ and so p^n is not a prime.

3. In any event, we need that $|E(K)|$ is divisible by a large prime. This requires, for all E and finite K, a method for computing $|E(K)|$: there are efficient (but complicated) algorithms for this.

Advantages of using Elliptic curves in cryptography

1. may choose values of curve coefficients, may choose finite field: this gives us a very large number of possible groups for Group Law
2. finding E and K with strong cryptographic properties is relatively easy
3. curves $E(K)$ with strong cryptographic properties much more secure than, say, $a \mapsto g^a \mod p$ in terms of the bit-size of p
4. thus we may decrease size of p when using Elliptic Curves, without losing security when compared to operating in multiplicative group $(\{1, 2, \ldots, p-1\}, \lambda x \lambda y \colon x \cdot y \mod p, 1)$ with suitable generator g

Above, $\lambda x \lambda y \colon x \cdot y \mod p$ represents multiplication modulo p.

Efficient computation with $E(K)$

Formulas for group law in $E(K)$ require many divisions if we use the K^2 representation, for example when $x_1 \neq x_2$ and λ equals

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

The latter expression really means $(y_2 - y_1) \cdot (x_2 - x_1)^{-1}$ in field K. This involves a multiplicative inverse in the finite field (expensive).

Solution: Use projective coordinates (X, Y, Z) instead of (X, Y) for computations. One good method is to transform $(0, 1, 0)$ back to $(0, 1)$, and (X, Y, Z) back to $(X/Z^2, Y/Z^3)$ for $Z \neq 0$.

Note that the latter also involves division but this is for the final transformation not for many intermediate computation steps.

This method transforms the long Weierstrass form of (5Smart) into

$$E : \; Y^2 + a_1XYZ + a_2YZ^4 = X^3 + a_2X^2Z^2 + a_4XZ^4 + a_6Z^6$$

Efficient storage with $E(K)$

EXERCISES:

1. For large primes p, let α be in $\mathbb{F}_p^*$ such that $\beta^2 \equiv \alpha \mod p$ has a solution. Then there are two such "square roots" β and $-\beta$.

   (a) Show that the numbers $\beta \mod p$ and $-\beta \mod p$ have different parities.

   (b) Let (x, y) be a point on $E(\mathbb{F}_p)$ for E of form (9). Let b be $y \mod 2$, the parity bit of y. Show that we can reconstruct, in principle, y from x, $E(\mathbb{F}_p)$, and b.

   Hint: You may assume an efficient algorithm for computing square roots of numbers that have such roots, for example Shank's algorithm. The details of such an algorithm don't interest us in this exercise.

2. Let $p = 3 \mod 4$ and a in $\mathbb{F}_p^*$ such that $\beta^2 \equiv a \mod p$ has a solution. Show that one such solution is $a^{(p+1)/4} \mod p$. In particular, explain why this number is well defined.

   In particular, in this case we don't need to run something like Shank's algorithm at all.

   Hint: Recall and use Fermat's Little Theorem, that $a^{p-1} \mod p = 1$.

Exercise

(Exercise 51 in exercise sheet) (part of it) Consider the Elliptic Curve

$$E : Y^2Z = X^3 + XZ^2 + 3Z^3$$

and the shorter form

$$E' : Y^2 = X^3 + X + 3$$

1. Let K be any finite field. Show that for all $(X, Y, Z) \in \mathbb{P}^2(K)$ with $Z \neq 0$ we have that

$$(X, Y, Z) \in E(K) \qquad \Rightarrow \qquad (X/Z, Y/Z) \in E'(K)$$

2. Show that $(4, 6)$ is in $E'(\mathbb{F}_7)$.

3. Show that $(1, 5, 2)$ is in $E(\mathbb{F}_7)$.

4. Show that $(4, 6)$ equals $(1/2, 5/2)$ over $\mathbb{F}_7$.

Moduli of special form

Recall the prime for `Secp256k1`:

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

This value is of form $b^t - a$ where a is "small". Value b is of form 2^w where w is the *word size* of the architecture. For the above p and setting $w = 64$ for modern desktop machines, this gives us $p = b^4 - a$ where $a = 4,296,968,273$.

Computing $u \cdot v \mod p$ where $0 \leq u, v < p$ involves first computing $x = u \cdot v$ with standard techniques.

But then we need fast modulo reductions for $x \mod p$, as in the following Algorithm II.1. from the reference given two slides from here:

INPUT: integer x
OUTPUT $r = x \mod p$
1. $q_0 = \lfloor x/b^t \rfloor$; $r_0 = x - q_0 \cdot b^t$; $r = r_0$; $i = 0$;
2. WHILE $q_i > 0$ DO
3. $\quad q_{i+1} = \lfloor q_i \cdot a/b^t \rfloor$; $r_{i+1} = q_i \cdot a + q_{i+1} \cdot b^t$;
4. $\quad i = i + 1$; $r = r + r_i$;
5. WHILE $r \geq p$ DO $r = r - p$
6. RETURN r

More efficient, and more complicated, methods exist when a above has few non-zero bits in its binary representation, e.g. for the p used in Bitcoin.

Summary of Elliptic Curves

- give us a finite abelian group through *Group Law*
- we seek representations of that group that make multiplication (the Group Law $+$) and exponentiation ($\star$) easy as operations
- but taking a discrete logarithm should be computationally hard
- also should be possible to generate random elements of that group with nearly uniform distribution (WHY?)
- elliptic curve $E(\mathbb{F}_q)$ believed to have similar level of security as cyclic group $(\{1, 2, \ldots, 1-p\}, \lambda(x, y) \colon x * y \mod p, 1)$ where $q \sim \sqrt[3]{p}$: *efficiency gains!*

We did not cover important computational aspects of *implementing* Elliptic Curves here, for example:

- how to compute $k \star G$ efficiently in $\ln(k)$
- how to shield against information flow in such computations (e.g. timing attacks)
- efficient representations of finite fields, e.g. for fields with characteristic 2 when field elements are polynomials modulo an irreducible polynomial

Advanced further reading on Elliptic Curves

If you want to find out more about this subject, beyond the scope of this course, I very much recommend:

Ian Blake, Gadiel Seroussi, Nigel Smart.
Elliptic Curves in Cryptography.
London Mathematical Society.
Lecture Notes 265.
Cambridge University Press, 1999.

Chapter 19 of Nigel Smart's book: *Secret Sharing Schemes*

EXAMPLE:

Secret s is a nuclear launch code.

Set of parties $\mathcal{P} = \{P, V, S, G\}$ where

P = President $\quad$ V = Vice-President $\quad$ S = Secretary of State $\quad$ G = General

The Problem: Split up secret s amongst parties such that only specific subsets of parties can recover the secret s.

For example, president and general should be able to recover secret. And vice-president, secretary of state, and general should be able to do this as well.

Each party A will be given a *share* s_A of secret s.

QUESTION: How can we solve this problem securely? Efficiently?

Monotone access structure

Intuition: If $\{P, G\}$ can recover secret, then any superset – for example $\{P, S, G\}$ – should also be able to recover the secret.

Definition 19.1: Let $\Gamma \subseteq \mathbb{P}(\mathcal{P})$ be a non-empty collection of subsets of a finite set $\mathcal{P}$. Then Γ is a *monotone access structure* if

1. $\mathcal{P}$ is in Γ, and so all parties together can recover the secret
2. If $A \subseteq B \subseteq \mathcal{P}$ and A is in Γ, then B is in Γ as well – closure under supersets.

Let $m(\Gamma)$ be the set of minimal elements of Γ with respect to subset inclusion.

Example of monotone access structure

$$\begin{aligned} \mathcal{P} &= \{P, V, S, G\} \\ \Gamma &= \big\{\{P, G\}, \{V, S, G\}, \{P, V, G\}, \{P, S, G\}, \{P, V, S, G\}\big\} \\ m(\Gamma) &= \big\{\{P, G\}, \{V, S, G\}\big\} \end{aligned}$$

Definition 19.2: S = set of secrets, $\mathcal{P}$ set of parties, Γ monotone access structure over $\mathcal{P}$. A secret sharing scheme is a pair of algorithms

1. *Share*(s, Γ): given secret s in S, this algorithm computes shares s_A of s for each party A in $\mathcal{P}$
2. *Recombine*(H): Let $H = \{s_A \mid A \in \mathcal{O}\}$ for some $\mathcal{O} \subseteq \mathcal{P}$. If $\mathcal{O}$ is in Γ, this algorithm should return s; otherwise it should return `nil`.

QUESTION: When should we consider (Share, Recombine) to be secure?

Security and Threshold Structures

Want *information-theoretic* security:

For all $\mathcal{O}$ *not* in Γ, the set of shares $\{s_A \mid A \in \mathcal{O}\}$ should not reveal *any* information about secret s.

For example, from set $\{s_V, s_G\}$ an attacker should learn no information about nuclear launch code s.

Threshold Structures: Form an important class of access structures. Let $\mathcal{P} = \{A_1, A_2, \ldots, A_n\}$ and $1 \leq t \leq n$. The *t-out-of-n* monotone access structure Γ is defined by

$$m(\Gamma) = \{\mathcal{O} \mid \mathcal{O} \subseteq \mathcal{P}, |\mathcal{O}| = t\}$$

EXAMPLE: the 2-out-of-4 structure satisfies

$$m(\Gamma) = \big\{\{A_1, A_2\}, \{A_1, A_3\}, \{A_1, A_4\}, \{A_2, A_3\}, \{A_2, A_4\}, \{A_3, A_4\}\big\}$$

Monotone access structure as Boolean formula

Idea: represent set $m(\Gamma)$ as formula in Disjunctive Normal Form (DNF):

$$m(\Gamma) \sim \bigvee_{\mathcal{O} \in m(\Gamma)} \bigwedge_{A \in \mathcal{O}} A \tag{12}$$

EXAMPLE: we may represent the monotone access structure for nuclear launch codes as

$$(P \wedge G) \vee (V \wedge S \wedge G) \tag{13}$$

Truth of formula in (12) exactly captures that secret s can be recovered.

Next, we use these formulas to derive first secret sharing scheme that is

- secure
- but not very efficient

Ito-Nishizeki-Saito Secret Sharing

EXAMPLE: We use the formula $(P \wedge G) \vee (V \wedge S \wedge G)$ from (13). Let secret s have l bits. All shares s_i have l bits as well.[3]

In (13), interpret $\wedge$ as exclusive-or $\oplus$ and $\vee$ as string concatenation $||$:

$$\begin{aligned} P \wedge G \sim \{P, G\} \in m(\Gamma) \sim s_1 \oplus s_2 &= s \\ V \wedge S \wedge G \sim \{V, S, G\} \in m(\Gamma) \sim s_3 \oplus s_4 \oplus s_5 &= s \end{aligned}$$

where equations $=$ need to be satisfied by these s_i. Share of agent A is

$$s_A = \text{concatenation of all } s_i \sim A \text{ occurring in some } \mathcal{O} \in m(\Gamma)$$

EXAMPLE:

- $s_P = s_1$ since P only occurs in $\{\mathbf{P}, G\} \sim \mathbf{s_1} \oplus s_2$
- $s_V = s_3$ since V only occurs in $\{\mathbf{V}, S, G\} \sim \mathbf{s_3} \oplus s_4 \oplus s_5$
- $s_S = s_4$ for similar reasons, but
- $s_G = s_2 \,||\, s_5$ since G occurs in both $\{P, \mathbf{G}\} \sim s_1 \oplus \mathbf{s_2}$ and $\{V, S, \mathbf{G}\} \sim s_3 \oplus s_4 \oplus \mathbf{s_5}$

[3]So an attacker would know that s has l bits. We may say that such public information about scheme parameters does not compromise information-theoretic security, although it may contribute to such compromise if a scheme is weak.

This scheme is inefficient but secure

1. fresh set of (sub)shares s_i for each set $\mathcal{O}$ in $m(\Gamma)$
2. some parties require longer shares, for example s_G
3. t-out-of-n structure has $\binom{n}{t}$ many sets in Γ, resulting in $\prod_{i=1}^{t-1}(n-i)$ many parts s_i for each share s_A

But this scheme is secure provided that shares s_i are generated independently and uniformly at random. Specifically, for

$$s = s_1 \oplus s_2 \tag{14}$$
$$s = s_3 \oplus s_4 \oplus s_5 \tag{15}$$

generate s_1 randomly, and set $s_2 = s \oplus s_1$; and generate s_3 and s_4 randomly and set $s_5 = s \oplus s_3 \oplus s_4$.

To an attacker, all s_i look random. For example, if attacker knows $s_S = s_4$ and $s_G = s_2 \,||\, s_5$, he only knows s_2 in (14) so this is like a one-time-pad.

Similarly, he only knows $s_4 \oplus s_5$ in (15) and so this functions like a one-time pad as well.

EXERCISE: **(Exercise 29.1 in exercise sheet)** compute Ito-Nishizeki-Saito shares for 2-out-of-4 structure over $\mathcal{P} = \{A, B, C, D\}$.

Replicated Secret Sharing Scheme

Definition: A subset $\mathcal{O}$ of $\mathcal{P}$ is *maximally non-qualifying* if

1. for all $\mathcal{O}'$ we have that $\mathcal{O} \subset \mathcal{O}'$ implies $\mathcal{O}' \in \Gamma$
2. $\mathcal{O} \notin \Gamma$

where $\subset$ denotes *strict* subset inclusion. Define

$$\mathcal{B} = \{\mathcal{P} \setminus \mathcal{O} \mid \mathcal{O} \text{ maximally non-qualifying}\} \tag{16}$$

Share generation works in two phases:

1. for each B_i in $\mathcal{B}$, generate a share s_i such that $s = \bigoplus_{B_i \in \mathcal{B}} s_i$
2. for each P in $\mathcal{P}$, share s_P is concatenation of all s_i for which P is in B_i

Example of Relicated Secret Sharing

Recall $\mathcal{P} = \{P, V, S, G\}$ for nuclear launch codes, where $m(\Gamma) = \big\{\{P, G\}, \{V, S, G\}\big\}$.

1. Compute all maximally non-qualifying sets:

$$\mathcal{O}_1 = \{P, V, S\} \qquad \mathcal{O}_2 = \{V, G\} \qquad \mathcal{O}_3 = \{S, G\}$$

   Note that $\mathcal{O}_1$ is neither a subset nor a super-set of any set in $m(\Gamma)$.

2. Compute complements of the sets in previous item:

$$B_1 = \mathcal{P} \backslash \{P, V, S\} = \{G\} \qquad B_2 = \mathcal{P} \backslash \{V, G\} = \{P, S\} \qquad B_3 = \mathcal{P} \backslash \{S, G\} = \{P, V\}$$

3. Generate random shares s_1, s_2, s_3 such that $s = s_1 \oplus s_2 \oplus s_3$.

4. Generate shares for each party:

$$\begin{array}{rcll} s_P & = & s_2 \,||\, s_3 & \text{since } P \text{ is in } (B_2 \cap B_3) \setminus B_1 \\ s_V & = & s_3 & \text{since } V \text{ is in } B_3 \setminus (B_1 \cup B_2) \\ s_S & = & s_2 & \text{since } S \text{ is in } B_2 \setminus (B_1 \cup B_3) \\ s_G & = & s_1 & \text{since } G \text{ is in } B_1 \setminus (B_2 \cup B_3) \end{array}$$

For example, why can parties from set $\{P, S\}$ not infer any information about secret s?

The attacker, that is parties P and S sharing their information, knows:

1. $s_P = s_2 \,||\, s_3$
2. $s_S = s_2$
3. s, which they know equals $s_1 \oplus s_2 \oplus s_3$

So attacker knows s_2 and s_3, and therefore knows $s_2 \oplus s_3$ in equation

$$s = s_1 \oplus (s_2 \oplus s_3) \tag{17}$$

ASSUMPTION: shares s_i are generated truly randomly, except for a single share (say s_3) where we set

$$s_3 = (s_1 \oplus s_2) \oplus s$$

But attacker can learn noting about s from this, as s_3 is the result of a one-time pad applied to plaintext s and random key $s_1 \oplus s_2$. So the equation in (17) means that s_1 and s are completely unknown and $s_2 \oplus s_3$ known but random. This functions like a one-time pad again.

Correctness and Information-Theoretic Security of Replicated Secret Sharing

Two parts to this:

(1) All $\mathcal{O}$ in Γ can recover secret s.

(2) No $\mathcal{O}$ in $\mathbb{P}(\mathcal{P}) \setminus \Gamma$ can recover secret s or learn anything about it.

Proof of (1): Let $\mathcal{O}$ be in Γ. Need to show that $H = \{s_A \mid A \in \mathcal{O}\}$ can recover secret s. Recall that $s = s_1 \oplus \cdots \oplus s_k$ for $\mathcal{B} = \{B_1, \ldots, B_k\}$ where $\mathcal{O}_i = \mathcal{P} \setminus B_i$ are maximal non-qualifying sets.

We show that H "contains" all s_1, s_2, $\ldots$, s_k. Let $1 \leq i \leq k$. Then $\mathcal{O}_i \notin \Gamma$ and so $\mathcal{O} \not\subseteq \mathcal{O}_i$ since $\mathcal{O}$ is in Γ and Γ is monotone.

Therefore, there is some A in $\mathcal{O}$ with $A \notin \mathcal{O}_i$, and so $A \in \mathcal{P} \setminus \mathcal{O}_i = B_i$.

By definition of s_A, it therefore "contains" the s_i. Since A is in $\mathcal{O}$, we know that s_A is in H. Thus, since i was arbitrary above, H contains all s_i and so parties in $\mathcal{O}$ can recover s as $s_1 \oplus \cdots \oplus s_k$.

Obviously, an implementation needs to guarantee that parties A know which of its shares come from B_i.

(2) No $\mathcal{O}$ in $\mathbb{P}(\mathcal{P}) \setminus \Gamma$ can recover secret s or learn anything about it.

Now let $\mathcal{O}$ not be in Γ. Need to show: from $H = \{s_A \mid A \in \mathcal{O}\}$ we cannot learn anything about the secret s.

Since $\mathcal{O} \notin \Gamma$, there either is some A' in $\mathcal{P}$ for which $\mathcal{O} \cup \{A'\}$ is not in Γ or $\mathcal{O}$ is maximally non-qualifying. Therefore, we can keep adding such elements A', A'', ... to $\mathcal{O}$ until the result is maximally non-qualifying.

But then there is some maximally non-qualifying set $\mathcal{O}_i$ with $\mathcal{O} \subseteq \mathcal{O}_i$. In particular, $B_i = \mathcal{P} \setminus \mathcal{O}_i$ is in $\mathcal{B}$ and disjoint with $\mathcal{O}$.

From this, we infer that *no* party in $\mathcal{O}$ gets the share s_i, as this share is not present in any of the s_A in H. So even if we assume that from H all shares s_j with $i \neq j$ can be inferred, the equation

$$s = s_i \oplus \bigoplus_{i \neq j} s_j$$

means that this looks like a one-time pad as nothing about s is know, nothing about s_i, and (all but one of) the s_i are generated uniformly at random.

Discussion

1. Want secure sharing schemes that are also *efficient*.
2. Want ability to recover secret if some shares are corrupted due to transmission errors, or even due to malicious tampering.
3. May want ability to recover secret even if some shares are maliciously corrupted.
4. Or at least we want to be able to detect transmission errors or malicious actions.

Such considerations lead to discussion of "codes" from Electrical Engineering for t-out-of-n schemes.

Requires some background from *finite fields*, which we covered already for Elliptic Curves above.

Reed-Solomon Codes

Error-correcting codes used in CD/DVD, BluRay, WiMAX, and in space missions such as the famous interstellar mission Voyager.

Mathematical setting:

- $q = p^n$ a prime power
- $\mathbb{F}_q$, the unique finite field with q many elements
- $\mathbb{X} \subset \mathbb{F}_q$, a set of size n – we assume that 0 is not in $\mathbb{X}$

Different elements of $\mathbb{X}$ represent different parties.

The *set of code words* is [4] is

$$\mathbb{P} = \{\sum_{i=0}^{t} f_i \cdot X^i \mid f_i \in \mathbb{F}_q\}$$

Parameter t in above sum is related to *error correction*.

Elements of $\mathbb{P}$ are polynomials of degree $\leq t$ over $\mathbb{F}_q$.

$|\mathbb{P}| = q^{t+1}$, since each f_i could independently have $|\mathbb{F}_q| = q$ values

[4] Note the overloading of the symbol $\mathcal{P}$ in Smart's book; it also denotes a set of parties. But context will always determine the proper meaning.

Actual code words:

Evaluate code word f in $\mathbb{P}$ at all points in $\mathbb{X} = \{x_1, x_2, \ldots, x_n\}$:

$$C = \{\big(f(x_1), f(x_2), \ldots, f(x_n)\big) \mid f \in \mathbb{P}\}$$

Note:

- representation of code words uses a linear ordering[5] on $\mathbb{X}$.
- code word $\big(f(x_1), f(x_2), \ldots, f(x_n)\big)$ has length $n \cdot \log_2 q$ but represents $(t+1) \cdot \log_2 q$ bits of information
- the latter gives us important *redundancy* for error correction when $t < n$

[5] Also know as a *total* ordering in the technical literature.

Example code word

Let us specify

$$\begin{aligned} q &= 101 \\ t &= 2 \\ n &= 7 \\ \mathbb{X} &= \{1, 2, 3, 4, 5, 6, 7\} \end{aligned}$$

In particular, q is a prime number and 0 is not in $\mathbb{X}$. Our data is f in $\mathbb{P}$ with

$$f = 20 + 57X + 68X^2$$

We transform this data into a code word

$$\begin{aligned} c &= (f(1) \mod q, f(2) \mod q, \ldots, f(7) \mod q) \\ &= (145 \mod 101, \ldots, 3751 \mod 101) \\ &= (44, 2, 96, 23, 86, 83, 14) \end{aligned}$$

We may now transmit or store code word c as a representation of f.

Data Recovery

QUESTION: How can we recover f from code word c?

Claim: if $t < n$ and no transmission error occurred in communication of c, we can *fully recover* f from c.

Note: if suffices to show that we can fully recover f_0, f_1, ..., f_t for

$$f = \sum_{i=0}^{t} f_i \cdot X^i$$

Receiver of code word c has n linear equations, one for each x_i in $\mathbb{X}$:

$c_i = f(x_i)$ (c_i and x_i are constants, $f_0, \ldots, f_t$ are variables)

Since $t < n$, we get $t + 1 \leq n$. So we can solve this system over $\mathbb{F}_q$.

That solution is unique since f is determined by $t + 1$ points: *Fundamental Theorem of Algebra*.

Easier method: Lagrange interpolation

No need to solve n equations, use *Lagrange interpolation* instead: define polynomials [6] $\delta_i(X)$ for all $1 \leq i \leq n$:

$$\delta_i(X) = \prod_{x_j \in \mathbb{X}, i \neq j} \frac{X - x_j}{x_i - x_j}$$

Note that these have the following properties:

1. $\delta_i(x_i) = 1$
2. $\delta_i(x_j) = 0$ for all $i \neq j$
3. $\delta_i(X)$ has degree $n - 1$
4. $\frac{X - x_j}{x_i - x_j}$ means $(X - x_j) \cdot (x_i - x_j)^{-1}$ where the latter is a multiplicative inverse in $\mathbb{F}_q$

Define $g(X) = \sum_{i=1}^{t} c_i \cdot \delta_i(X)$. Then:

(i) $g(x_i) = c_i$ by items 1. and 2. above, for all $1 \leq i \leq n$

(ii) degree of g is $\leq n - 1$ by item 3. above

(iii) g is unique with respect to (i) and (ii) above

(iv) but then g equals f as $t + 1 \leq n$

We apply this now to secret sharing.

[6] The formal variable $\mathbb{X}$ in $\delta_i(X)$ is of course different from $\mathbb{X}$ as our chosen set of points; but I prefer to stick to this overload of notation as it is used in Smart's book.

Shamir Secret Sharing

$(t+1)$-out-of-n secret sharing scheme

Security property: t or less than t parties, when colluding, should not learn anything about the secret, which is an element s in field $\mathbb{F}_q$.

Trusted dealer computes and distributes shares of secret s as follows:

1. generates random integers f_i in $\mathbb{F}_q$ for all $1 \leq i \leq t$
2. constructs f in $\mathbb{P}$ where $f = s + \sum_{i=1}^{t} f_i \cdot X^i$
3. each party is identified with a unique element in $\mathbb{X}$, so $|\mathbb{X}| = n$
4. party i in $\mathbb{X}$ is given the share $s_i = f(i)$ of secret s

Typically, $\mathbb{X}$ equals $\{1, 2, \ldots, n\}$. The Reed-Solomon code word for secret s is then $(s_1, s_2, \ldots, s_n)$.

For general $\mathbb{X}$, we only need a linear order[7] on $\mathbb{X}$ to represent code word as a vector.

[7]Some of you may know this as a *total* order, which has the same meaning.

Shamir Secret Sharing: secret recovery

Any subset of $\{s_1, s_2, \ldots, s_n\}$ of size $\geq t+1$ allows us to recover f via Lagrange interpolation.

Once that is done, we may recover secret s by computing $f(0)$.

More efficient secret recovery: Compute $s = f(0)$ directly without computing formal polynomial f.

For $\mathbb{Y} \subseteq \mathbb{X}$, we set

$$\begin{aligned} r_{x_i,\mathbb{Y}} &= \prod_{x_j \in \mathbb{Y}, x_j \neq x_i} \frac{-x_j}{x_i - x_j} \\ r_{\mathbb{Y}} &= (r_{x_i,\mathbb{Y}})_{x_i \in \mathbb{Y}} \qquad \text{(recombinant vector)} \end{aligned}$$

For any $\mathbb{Y}$ with $|\mathbb{Y}| \geq t+1$, we can then recover secret s as

$$s = \sum_{x_i \in \mathbb{Y}} r_{x_i,\mathbb{Y}} \cdot s_i$$

Example of share distribution

- q is the prime 101, $n = 7$, $t = 2$, the secret s equals 87
- $\mathbb{F}_q = \{0, 1, 2, \ldots, 100\}$, computations modulo 101
- dealer generates two random elements of $\mathbb{F}_q$:

$$f_1 = 29 \qquad\qquad f_2 = 71$$

- for $f = 87 + 29X + 71X^2$
- representation of set of parties: $\mathbb{X} = \{1, 2, 3, 4, 5, 6, 7\}$
- share for each i is $f(i)$:

$$\begin{aligned} s_1 &= f(1) \equiv 187 \equiv 86 \mod 101 \\ &\ldots \\ s_7 &= f(7) \equiv 87 + 29 \cdot 7 + 71 \cdot 7^2 \\ &\equiv 87 + 1 + 45 = 133 \equiv 32 \mod 101 \end{aligned}$$

So $s_1 = 86$ and $s_7 = 32$. We similarly compute the other shares as

s_2	s_3	s_4	s_5	s_6
25	5	26	88	90

Example of secret recovery

We illustrate this in the version that uses the *recombinant vector* r_Y.

Recall that fractions $\frac{a}{b}$ denote $a \cdot b^{-1}$ where b^{-1} is multiplicative inverse in $\mathbb{F}_q$, which exists for all $b \neq 0$.

Suppose that $\mathbb{Y} = \{2, 4, 6\}$ and so $|\mathbb{Y}| = 3 \geq t + 1 = 3$.

1. We compute $r_y = (r_{2,\mathbb{Y}}, r_{4,\mathbb{Y}}, r_{6,\mathbb{Y}})$ as follows:

$$\begin{aligned}
r_{2,\mathbb{Y}} &= \big((-4) \cdot (2-4)^{-1}\big) \cdot \big((-6) \cdot (2-6)^{-1}\big) \\
&\equiv (-4) \cdot 50 \cdot (-6) \cdot 25 \equiv 3 \mod 101 \\
r_{4,\mathbb{Y}} &= \big((-2) \cdot (4-2)^{-1}\big) \cdot \big((-6) \cdot (4-6)^{-1}\big) \\
&\equiv (-2) \cdot 51 \cdot (-6) \cdot 50 \equiv 98 \mod 101 \\
r_{6,\mathbb{Y}} &= \big((-2) \cdot (6-2)^{-1}\big) \cdot \big((-4) \cdot (6-4)^{-1}\big) \\
&\equiv (-2) \cdot 76 \cdot (-4) \cdot 51 \equiv 1 \mod 101
\end{aligned}$$

2. We recover secret $s = 87$ as

$$\begin{aligned}
\sum_{i \in \mathbb{Y}} r_{i,\mathbb{Y}} \cdot s_i &\equiv 3 \cdot 25 + 98 \cdot 26 + 1 \cdot 90 \\
&\equiv 75 + 23 + 90 \\
&\equiv 87 \mod 101 \\
&= 87
\end{aligned}$$

Security of Shamir Secret Sharing

Suppose we have $\mathbb{Y}$ with $|\mathbb{Y}| \not\geq t+1$, i.e. $|\mathbb{Y}| \leq t$. We claim that the agents $(A_i)_{i\in\mathbb{Y}}$ cannot learn *anything* about secret s when combining their shares $(s_i)_{i\in\mathbb{Y}}$.

By monotonicity, it suffices to show this for the case when $|\mathbb{Y}| = t$: this is the most powerful scenario from the perspective of agents A_i with i in $\mathbb{Y}$.

From their combined shares $(s_i)_{i\in\mathbb{Y}}$, the agents can compute a unique polynomial g of degree $t-1$. Of course, it is extremely unlikely that $g(0) = s$. But even it this occurred, the scheme is *information-theoretically secure*:

For all x in $\mathbb{F}_q$, we use Lagrange Interpolation to infer that there exists a polynomial f of degree t such that

$$\begin{array}{rcll} f(0) & = & x & \\ f(x_i) & = & g(x_i) = s_i & \text{(for all } i \text{ in } \mathbb{Y}) \end{array}$$

In other words, any element of $\mathbb{F}_q$ is a possible recovered secret based on the shares in $(s_i)_{i\in\mathbb{Y}}$; these agents can therefore not learn anything about s.

Error Detection and Correction

Code word $c = (s_1, s_2, \ldots, s_n)$ of secret shares.

QUESTION: what if some s_i are errors? For example, if $f(2) \neq s_2$?

Error detection: Ability to determine whether c is free of errors. If errors do occur, can we locate them?

Error correction: Ability to (detect and) correct a certain number of errors.

Reality check: May not be able to recover from too many errors.

coding theory	cryptography
errors are "passive":	errors are "active":
random noise	adversary interacts to compromise
or random failures	information security
	e.g. confidentiality or integrity

Error detection for Reed-Solomon code

For code word $c = (c_1, c_2, \ldots, c_n)$ we already saw how to reconstruct f as

$$g = \sum_{i=1}^{n} c_i \cdot \delta_i(X)$$

Degree of constructed g is $\leq t$ and equal to degree of f.

So if the degrees of g and f are different, e.g. if degree of g is $> t$, we know that there are errors. And errors are likely to give us such different degrees.

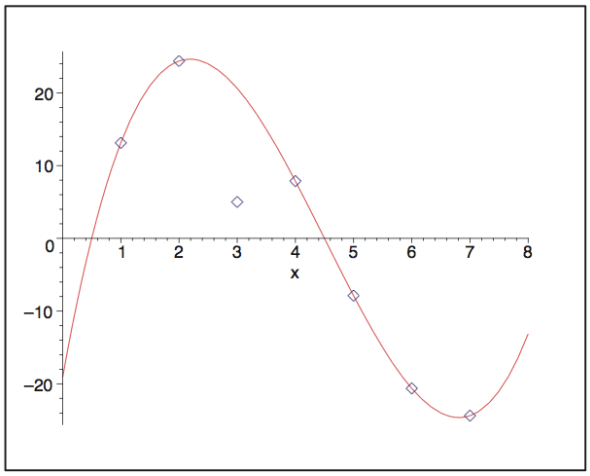
Thus we can detect errors with "high probability".

QUESTION: Once errors are detected, how can we locate and correct them? And how many errors can we correct?

ASSUMPTION: There are at most $e \geq 0$ errors in code word $c = (c_1, c_2, \ldots, c_n)$.

Example

Figure 19.2 from Smart's book on page 410: six interpolation points of cubic function with one error ($e = 1$):



Idea: take $S \subset \mathbb{X} = \{1, 2, 3, 4, 5, 6\}$ a *strict* subset of $\mathbb{X}$.

For $e = 1$ let $|S| = 5$. If g reconstructed from S has degree 3, then likely that S is *"valid"* subset; otherwise, S contains at least one error.

We only need to find one "valid" set S to succeed. But there are $\frac{n!}{e! \cdot (n-e)!}$ many such sets to explore in the worst case.

Moreover, what if we knew that there are additionally $s > 0$ *erasures* in the received code word?

Dealing with errors and erasures

Need $t < n - e - s$ then to recover polynomial of degree t from a "valid" set.

However, this polynomial *may not be unique*! For example, when $t = n - e - s + 1$, then all S of "search size" will construct polynomial of degree t.

Fact: (won't show this) Error correction can be done uniquely if

$$n > t + 2e + s \qquad (*)$$

QUESTION: How do we correct errors efficiently under the assumption (*) above?

Berlekamp-Welch algorithm

Preconditions:

1. There are s missing values in code word c.
2. Number of errors e is bounded by $e < t < \frac{n-s}{3}$.

So we are given $n - s$ supposed values $y_i = f(x_i)$ and so know the values in set $\{(x_i, y_i) \mid 1 \leq i \leq n - s\}$. And we know that at most e of these value pairs are wrong.

Idea: Construct *bivariate* polynomial over $\mathbb{F}_q$:

$$Q(X, Y) = f_0(X) - f_1(X) \cdot Y \qquad \text{where}$$

(1) $f_1(0) = 1$

(2) $degree(f_0) \leq 2t$

(3) $degree(f_1) \leq t$

The number of coefficients in Q, which we think of as *variables* to compute, is at most

$$(2t + 1) + (t + 1) - 1 = 3t + 1$$

which follows from items (1)-(3) above, the -1 term coming from item (1).

Berlekamp-Welch algorithm continued

From $\{(x_i, y_i) \mid 1 \leq i \leq n-s\}$ we get $n-s$ linear equations $Q(x_i, y_i) = 0$.

By assumption, $t < \frac{n-s}{3}$ and so $3t < n-s$, implying that $3t+1 \leq n-s$.

But $3t+1 \leq n-s$ ensures that the solution to the above $n-s$ linear equations reconstructs Q, i.e. the $3t+1$ coefficients of f_0 and f_1.

Next, use *Extended Euclidean Algorithm* over the ring of polynomials over $\mathbb{F}_q$ to compute

$$f = \frac{f_0}{f_1} \tag{18}$$

Claim: Equation (18) holds for the f that was used to generate the secret sharing instance.

Intuition: For the original f used to construct the code word, we then have $f \cdot f_1 = f_0$ which also gives us that

$$Q(X, f(X)) = f_0(X) - f_1(X) \cdot f(X) = f_0(X) - f_0(X) = 0$$

Proof of claim (18)

Consider $P(X) = Q(X, f(X))$ where f is polynomial we want to reconstruct.

$degree(P(X)) \leq 2t$ since $degree(f_0) \leq 2t$ and $degree(f_1 \cdot f) \leq 2t$.

$P(X)$ has at least $n - s - e$ zeros, the number of valid pairs (x_i, y_i) in the received code word.

By assumption, $e < t < \frac{n-s}{3}$, so the number of zeros of $P(X)$ is at least

$$n - s - e > n - e - t > 3t - t = 2t$$

But this means that $P(X)$ has more zeros than its degree. Therefore, $P(X) \equiv 0$ is always zero, so $f_0 - f_1 \cdot f \equiv 0$. Since $f_1 \not\equiv 0$, this yields the claim (18):

$$f = \frac{f_0}{f_1}$$

Example

Revisit previous example with $n = 7$, $t = 2$, $q = 101$, $\mathbb{X} = \{1, 2, 3, 4, 5, 6, 7\}$, and $c = (44, 2, 96, 23, 86, 83, 14)$.

Assumptions:

1. No erasures occurred, i.e. $s = 0$.
2. One error occurred: 96 is received as 25, and so $e = 1$.

QUESTION: How many errors can we hope to correct here?

We have $2 = t < \frac{n-s}{3} = \frac{7-0}{3} = 2.\overline{3}$ which is true. Thus, $e < t < \frac{n-s}{3}$ holds when $e = 1$.

Therefore, we should be able to recover from the error in $\tilde{c} = (44, 2, \mathit{25}, 23, 86, 83, 14)$. Polynomial Q is

$$Q(X, Y) = f_{0,0} + f_{1,0}X + f_{2,0}X^2 + f_{3,0}X^3 + f_{4,0}X^4 - (1 + f_{1,1}X + f_{2,1}X^2) \cdot Y$$

Solve corresponding linear equations: see page 358 of Smart's textbook for details. Solution is

$$(f_{0,0}, f_{1,0}, f_{2,0}, f_{3,0}, f_{4,0}, f_{1,1}, f_{2,1}) = (20, 84, 49, 11, 0, 67, 0)$$

From this we have

$$\begin{aligned} f_0(X) &= 20 + 84X + 49X^2 + 11X^3 \qquad\qquad f_1(X) = 1 + 67X \\ \frac{f_0(X)}{f_1(X)} &= 20 + 57X + 68X^2 \qquad\qquad \text{(using Extended Euclid algorithm)} \end{aligned}$$

But this is f we had at the start of computing the code word $c = (44, 2, 96, 23, 86, 83, 14)$. So we recovered f and so can detect and correct the error 25 to 96. Let's verify (18):

$$\begin{aligned} f(X) \cdot f_1(X) &= (20 + 57X + 68X^2) \cdot (1 + 67X) \qquad (19) \\ &= 20 + (57 + 20 \cdot 67) \cdot X + (68 + 57 \cdot 67) \cdot X^2 + (68 \cdot 67 + 11) \cdot X^3 \\ &= 20 + 84X + 49X^2 + 11X^3 \\ &= f_0(X) \end{aligned}$$

Chapter 20 of Smart's book:
Commitments and Oblivious Transfer

Security protocols are wise to assume that their participants may not be honest.

Bitcoin's consensus mechanism does not trust participants, for example.

Aim: *Design protocols in which parties can only cheat by also exposing that they cheated.*

Note: Bitcoin does not expressly pursue this aim. For example, there is no point in producing a proof of work that won't verify.

We next present two primitives that detect cheating. These primitives are building blocks for more complicated protocols to be studied later on.

We first illustrate commitment schemes through a simple and familiar example.

Example of a Commitment Scheme: "paper-scissors-stone"

Alice and Bob simultaneously choose a value from set

$$\{paper, scissors, stone\}$$

Outcome of this play:

- paper wraps stone: paper wins
- stone blunts scissors: stone wins
- scissors cut paper: scissors win
- if Alice and Bob choose same value, the play ends in a tie

Question: *What if Alice and Bob want to play this game over the phone?* If both are dishonest, the party who announces his or her value first will lose!

Root of problem: *asynchronous* nature of play.

Solution: party who announces first, announces not the value but a commitment to that very value.

The commitment needs to be *binding* and *concealing*.

A realization through a hash function

Consider the following protocol run that Alice (A) and Bob (B) can use:

1. $A \longrightarrow B$: h_A defined as $H(R_A \,||\, paper)$
2. $B \longrightarrow A$: $scissors$
3. $A \longrightarrow B$: $R_A,\ paper$

where H is a hash function that A and B share. Note that

- A first commits to the value *paper*, the commitment is hash value h_A
- A uses a random value R_A to prevent B from "decoding" the commitment to its value *paper* from h_A
- B should only send value *scissors* once he has received commitment h_A – why?
- after step 3., B can verify that A really chose *paper* by checking that h_A equals $H(R_A \,||\, paper)$, the value received in step 3.

Security Properties of Commitment Schemes

Concealing Property of Commitment Scheme:
also known as *Hiding* Property:

> Bob won't know which value A has committed to, since Bob does not know R_A after step 1.

Binding Property of Commitment Scheme:

> after Step 2., Alice knows she lost the play but she is unable to cheat now, so she is *bound* by her commitment

Question: How could Alice cheat? Only *stone* beats *scissors*, so she needs to find some R'_A such that

$$H(R_A \mid\mid paper) = H(R'_A \mid\mid stone)$$

This would require her to break the second-preimage resistance of H.

Types of Security for Commitment Schemes

Recall two notions of security:

1. *information-theoretic security*: attacker with infinite computing power cannot break the scheme
2. *computational security*: attacker with specified computation resources (e.g. probabilistic polynomial time) cannot break the scheme

For example, we studied these notions for encryption. But now we have a scheme with *two* properties:

- *concealing:* protects the interests of the sender
- *binding:* protects the interests of the receiver

Need to study and reconcile security of *both* properties!

Formal definition of Commitment Schemes

A commitment scheme is a public algorithm C which takes a value x and some randomness r as input to produce a commitment

$$c = C(x, r)$$

Definition 20.1 (Binding): C is information-theoretically (respectively, computationally) binding if no attacker with infinite (respectively bounded) computing power can win the following 1-person game:

(B1) The attacker produces x and r, which give rise to her commitment $c = C(x, r)$.

(B2) The attacker must now find $x' \neq x$ and some r' such that

$$C(x, r) = C(x', r')$$

If the attacker can do this, she wins that game – and can now claim to have the commitment x' instead of x.

Formal definition of Concealing

Definition 20.2 (Concealing): C is information-theoretically (respectively, computationally) concealing if no attacker with infinite (respectively bounded) computing power can win the following *2-person* game:

(C1) The attacker produces two different messages x_0 and x_1 of equal length, and sends these to the challenger.

(C2) The challenger generates r at random, and generates a random bit b in $\{0, 1\}$.

(C3) The challenger computes $c = C(x_b, r)$ and passes c to the adversary.

(C4) The adversary needs to guess b correctly.

Note: Concealing involves *two* players. Wins are actually *repeated* plays with a probability of $0.5 + \epsilon$ for some $\epsilon > 0$ of guessing correctly on average.

First results on Commitment Schemes

Lemma 20.3: There is no commitment scheme that is *both* information-theoretically binding and information-theoretically concealing.

Proof: Let C be a commitment scheme that is information-theoretically binding. We claim that an attacker with infinite computing power can now win the Concealing game:

- Attacker generates all possible pairs (x, r) until $C(x, r) = c$ for the c that challenger passed to attacker in step (C3).
- By information-theoretic binding, the value x that attacker found must be in $\{x_0, x_1\}$ a set of size 2.
- So attacker guesses b with *certainty* as the unique b with $x = x_b$.

Result on hash-based commitment schemes

Lemma 20.4: For a hash function H, the commitment scheme

$$C(x, r) = H(r \mid\mid x)$$

is *"at best"* computationally binding and information-theoretically concealing.

"Proof:" We only know hash functions that are computationally secure against preimage resistance and second-preimage resistance. Above scheme C relies on second-preimage resistance to guarantee binding. So C is at best computationally binding.

Concealing property of C rests on preimage resistance of hash function H, suggesting only computational security.

But: If R chosen from large enough set, and if H has many (combinatorial) collisions, then $\lambda R\colon H(R \mid\mid x)$ gets close to being information-theoretically secure.

Note: If we treat $\lambda R\colon H(R \mid\mid x)$ as a *random oracle*, this would make C information-theoretically concealing.

Mathematical setting for other commitment schemes

Now study important commitment schemes that share a mathematical setting:

- G as a finite, commutative group of prime order q
- G generated by an element g
- h in G such that x with $g^x = h$ in G is unknown to all system users
- *but:* values of q, g, and h are *public*

Note: We may generate such groups using *Elliptic Curves*.

Some mathematical details

Let p and q be prime with $p = 2 \cdot q + 1$. Then p is called a *"safe prime"*.

Notation: $\mathbb{F}_p^* = \mathbb{F}_p \setminus \{0\}$ is a multiplicative, commutative group due to field axioms

Assumption: A cryptographic hash function $H : \mathbb{Z} \to \mathbb{F}_p^*$

Compute generator g by randomised algorithm:

repeat {
  $r = random(\mathbb{Z})$;
  $f = H(r)$;
  $g = f^2 \ \% \ p$; % is modulo operation, and note that $(p-1)/2 = q$
} *until* $(g \neq 1)$
return (r, g);

Note:

- Element g is random element of $\mathbb{F}_p^*$ of order q: Lagrange's Theorem and safe prime p means proper subgroups of $\mathbb{F}_p^*$ have order 1, 2 or q.
- Everyone can verify that g was generated from seed r – why does this matter?

Generation of h is similar

repeat {
  $r = random(\mathbb{Z})$;
  $f = H(r)$;
  $h = f^2$ % p; % is modulo operation, and note that $(p-1)/2 = q$
} *until* $(h \neq 1 \wedge h \neq g)$
return (r, h);

Note:

- h is also random element of $\mathbb{F}_p^*$ or order q, seed r verifies generation of h
- *but:* subgroups of $\mathbb{F}_p^*$ are cyclic and there is *unique* cyclic subgroup for every factor of $p-1$, including factor q
- therefore: h must be in subgroup of $\mathbb{F}_p^*$ generated by g
- as h is random such element, we don't know any z satisfying

$$h = g^z \mod p$$

Commitment Scheme based on ElGamal encryption

$$C(x, a) = E_a(x) = (g^a, x \cdot h^a)$$

- above all exponentiations are modulo p
- a is random integer modulo q
- x is an integer modulo q – the committed value

The commitment is revealed as the pair (a, x).

Unrealistically small, worked example

- let $p = 107 = 2 \cdot 53 + 1$ and $q = 53$ is prime
- generate g: we found $r = 248798$, suppose $H(r) = 26$, $26^2 \mod 107 = 34 \not\equiv 1$ and so $g = 34$; output is $(r, g) = (248798, 34)$
- generate h: we found $r = 8927591$, suppose $H(r) = 81$, then $81^2 \mod 107 = 34 = g \Rightarrow$ need to regenerate r; say $r = 512894$, suppose $H(r)$ is now 59, then $59^2 \mod 107 \equiv 57 (\not\equiv g, \not\equiv 1)$, so $h = 57$; output $(r, h) = (512894, 57)$

Of course, one could compute a z with $g^z = h \mod 107$ for such small parameters.

Alice generates her commitment: she chooses $a = 52$ randomly in $\mathbb{F}^*_{107}$ modulo 53. Her committed value x in $\mathbb{F}^*_{107}$ modulo 53 equals 27.

She computes

$$c = C(x, a) = E_a(x) = (g^a, x \cdot h^a) = (34^{52} \mod 107, 27 \cdot 57^{52} \mod 107) = (85, 23)$$

as her commitment.

Later, she can reveal this as $(a, x) = (52, 27)$ which Bob can use to verify c; the latter is possible since g, h, and p are public. Assuming that it is public knowledge that a safe prime is used, this makes q also public as $(p - 1)/2$.

Security properties of ElGamal Commitment Scheme

Lemma 20.6: The commitment scheme $E_a(x)$ is information-theoretically binding and computationally concealing.

Proof: $E_a(x)$ "is" ElGamal encryption with respect to public key h, and where $E_a(x)$ does not need associated private key (some z with $g^z = h \mod p$).

Computational concealing follows from *semantic security* of ElGamal encryption, which depends on G satisfying the decisional Diffie-Hellman assumption (we won't cover this topic of "semantic security" here).

Since decryption is correct for ElGamal, and so a ciphertext has a unique matching plaintext, the commitment scheme $E_a(x)$ is information-theoretically binding.

Note: the *structure* of this proof is important to us, we won't cover details of ElGamal encryption.

A commitment scheme with "dual" properties

$$C(x, a) = B_a(x) = h^x \cdot g^a$$

This is the *Pedersen* Commitment Scheme:

- again, a is random – a *blinding number*
- x is committed value
- all exponentiations are modulo p

Commitment is revealed as (a, x), the same as for scheme $E_a(x)$.

Question: In $B_a(x)$, why does a blind the value of x even to a computationally unbounded attacker?

Next, we prove that $B_a(x)$ has dual properties of $E_a(x)$.

Security properties of Pedersen's Commitment Scheme

Lemma 20.7: The commitment scheme $B_a(x)$ is information-theoretically concealing. If the committer does not know the discrete logarithm of h to base g, then $B_a(x)$ is computationally binding.

Proof: *Information-theoretically concealing:* let $c = B_a(x) = h^x \cdot g^a$. Prior to revelation of pair (a, x), the attacker examines all possible $\tilde{x}$ as candidate committed values.

For each $\tilde{x}$, there is some $\tilde{a}$ such that $c = h^{\tilde{x}} \cdot g^{\tilde{a}}$ since $g^{\tilde{a}}$ is $c \cdot (h^{\tilde{x}})^{-1}$ and c is in subgroup generated by g. This implies information-theoretic security for concealment.

Computationally binding: By Lemma 20.3 and the fact just shown, $B_a(x)$ is at best computationally binding. Under the assumption of Lemma 20.7, it suffices to show how to convert an efficient algorithm for breaking binding into an efficient algorithm for computing the discrete logarithm of h to base g:

Suppose attacker can output another pair (b, y) such that

$$h^y \cdot g^b = h^x \cdot g^a = c$$

Then we get $h^{y-x} = g^{a-b}$. Since h is of form g^z for unknown z, we see that $y - x$ divides $a - b$. Therefore,

$$z = \frac{a-b}{y-x}$$

gives us $g^z = h$. This is the claimed reduction of breaking binding of (a, x) with (b, y) to computing the discrete log of h to base g. Since the latter it assumed to be impossible, we infer that the former must be impossible.

Another example commitment scheme

The scheme

$$C(x, r) = B(x) = g^x$$

- is information-theoretically binding (it is deterministic),
- does not use blinding (since r is not used), and
- concealment relies on hardness of discrete logarithm problem in group G.

Note: both $B(x)$ and $B_a(x)$ have *homomorphic properties*:

$$\begin{aligned} B(x_1) \cdot B(x_2) &= B(x_1 + x_2) \\ B_{a_1}(x_1) \cdot B_{a_2}(x_2) &= B_{a_1+a_2}(x_1 + x_2) \end{aligned}$$

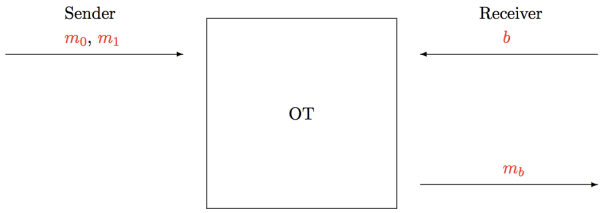
This property has many applications, for example in *electronic voting*: see e.g. Peter Ryan's recent work on *Selene* for improved *coercion resistance* in internet-based voting.

Oblivious Transfer (OT)

Oblivious $\cong$ Forgetful

A protocol between two distrusting parties, Figure 20.3 of Smart's book on page 422:

1. $S \longrightarrow OT$: m_0, m_1 (sender sends two messages)
2. $R \longrightarrow OT$: b (receiver chooses b in $\{0, 1\}$ as request for m_b)
3. $OT \longrightarrow R$: m_b (receiver receives requested message m_b)



Security Properties:

(i) Sender should not learn value of bit b.

(ii) Receiver should not learn value of m_{1-b}.

Requirement: Need to eliminate reliance on a third, trusted party OT.

A solution for messages of equal length

Mathematically, the setting is the same as the one studied for Commitment Schemes:

- $p = 2q + 1$ safe prime, i.e. p and q prime
- g in $\mathbb{Z}_p^*$ of order q: subgroup G of $\mathbb{Z}_p^*$ generated by g
- $H \colon \mathbb{Z}_p^* \to \{0,1\}^n$ cryptographic hash function
- h of form $g^x \mod p$ where x is secret; we will qualify meaning of "secret" below

ElGamal public/private key pair (h, x):

- h is public key
- x is private key

Can encrypt/decrypt n-bit messages (exponentiations all $\mod p$):

- given message m of n bits
- generate random k in $\mathbb{Z}_q$
- compute ciphertext $c = (c_1, c_2) = (g^k, m \oplus H(h^k))$

Exercise: Show that ciphertext (c_1, c_2) as above is *correctly* decrypted as $c_2 \oplus H(c_1^x)$.

Note: Anybody can produce ciphertexts since p, g, h, H are public.

Only those who know secret x can decrypt though – what assumption does this depend on?

ElGamal cryptosystem: good example for *hybrid* encryption

$$c = (c_1, c_2) = (g^k, m \oplus H(h^k))$$

- first component g^k is *key encapsulation mechanism* (KEM), k is ephemeral key
- second component $m \oplus H(h^k)$ is *data encryption mechanism* (DEM) – the "actual" ciphertext

We use this system in ingenious way to realise an oblivious transfer protocol, which also eliminates need for, and trust in, third party OT.

Key idea: Receiver has two public keys h_0 and h_1, but only knows private key for k_b when requesting message m_b.

ElGamal based OT protocol

1. Sender generates random c in G, such that Receiver won't be able to compute discrete logarithm of c to base g. Sender sends c to receiver.
2. Receiver generates random x in $\mathbb{Z}_q$ as private key and two public keys $h_b = g^x$ and $h_{1-b} = c \cdot h_b^{-1}$ for her choice of bit b. Receiver sends h_0 and h_1 to Sender.
3. Sender generates k_0 and k_1, and encrypts m_0 using h_0 and m_1 using h_1: $(g^{k_0}, m_0 \oplus H(h_0^{k_0}))$ and $(g^{k_1}, m_1 \oplus H(h_1^{k_1}))$, and sends both ciphertexts to Receiver.
4. Receiver uses x to decrypt $(g^{k_b}, m_b \oplus H(h_b^{k_b}))$

Note: Receiver knows private key for h_b, which is x, but does not know private key for h_{1-b} as she does not know the discrete logoarithm of c to base g.

Optimisation of this protocol

Figure on page 423 of Smart's book:

Sender		Receiver
$c \in G$	$\xrightarrow{c}$	
		$x \in \mathbb{Z}_q$,
		$h_b = g^x$,
	$\xleftarrow{h_0}$	$h_{1-b} = c/h_b$,
$h_1 = c/h_0$		
$k \in \mathbb{Z}_q$		
$c_1 = g^k$		
$e_0 = m_0 \oplus H({h_0}^k)$		
$e_1 = m_1 \oplus H({h_1}^k)$	$\xrightarrow{c_1, e_0, e_1}$	
		$m_b = e_b \oplus H({c_1}^x)$

- Sender knows c, so Receiver only has to send h_0 since Sender can compute $h_1 = c \cdot h_0^{-1}$ for c equals $h_{1-b} \cdot h_b$.
- We may assume $k_0 = k_1$ in both encryptions – why?

Security analysis of this OT protocol

What about the two security properties for oblivious transfer?

(1) Can the Sender guess the random bit b with probability > 0.5? He only learns h_0 from Receiver, and h_0 is random element in G.

(2) Can Receiver learn anything about m_{1-b}?

Receiver only learns random element c, and c_1, e_0, and e_1. So she does not know k either. But

$$m_{1-b} = e_{1-b} \oplus H(h_{1-b}^k)$$

Receiver knows first argument of $\oplus$; but cannot compute second argument if H is a *random oracle*: in that case, it would have to compute argument of H, the value of h_{1-b}^k.

Receiver knows $c_1 = g^k$ and knows h_{1-b} which is of form g^z for some z. Computing h_{1-b}^k then reduces to solving the *Diffie-Hellman* problem in G (believed to be hard for G such as the one chosen here):

$$\text{knowing } g^k \text{ and } g^z, \text{ find } y \text{ such that } y = g^{k \cdot z}$$

noting that $g^{k \cdot z} = (g^z)^k = h_{1-b}^k$.

Diffie-Hellman problem formally

Please see Definition 3.1 on page 52 of Smart's textbook:

(DHP) Given g in G and elements a and b in G of form

$$a = g^x \qquad\qquad b = g^y$$

where the values of x and y are not known, find c in G such that

$$c = g^{x \cdot y}$$

Recall Discrete Logarithm Problem (DLP), Figure 3.1 on page 51 of Smart's textbook: given g and h in G find x, if there exists one, such that $h = g^x$.

Exercise: Show that DHP is no harder than DLP.

Chapter 21 of Nigel Smart's book: *Zero-Knowledge Proofs*

Alice wants to convince Bob that she knows some x, without Bob finding out any information about x.

Examples:

- Alice is computer account user, Bob is authentication server, and x is the password of Alice.
- Alice is a millionaire, Bob is registrar of Millionaire's Club, and x is Alice's proof that she is entitled to membership – without disclosing her actual wealth.
- Alice is student, Bob is voting official, and x is proof that Alice voted in the student election – without revealing how she voted.

Question: How can we overcome this apparent contradiction and solve this?

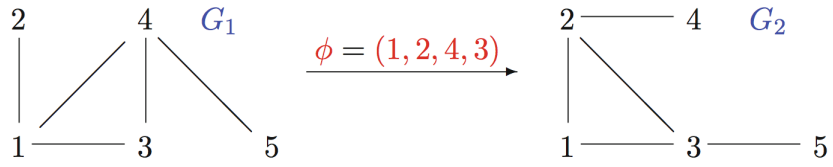
Important example: graph isomorphism

- $V = \{1, 2, \ldots, n\}$ set of vertices
- $E \subseteq V \times V$ *undirected edges*: $(i, j) \in E$ implies $(j, i) \in E$
- graph $G = (V, E)$

Graph G_1 *isomorphic* to G_2 iff

1. $V_1 = V_2$, and
2. $\exists$ bijection $\phi\colon V_1 \to V_2$ such that for all $(i, j) \in V_1 \times V_1$, we have $(i, j) \in E_1$ iff $(\phi(i), \phi(j)) \in E_2$.

Example from page 426 of Smart's book: G_1 and G_2 are isomorphic where $\phi(1) = 2$, $\phi(2) = 4$, $\phi(4) = 3$, $\phi(3) = 1$, and $\phi(i) = i$ for all other i, i.e. for $i = 5$.



Notation for Zero-Knowledge Proofs

Alice ≡ Prover ≡ "Peggy"

Bob ≡ Verifier ≡ "Victor"

Assumptions:

(i) graphs G_1 and G_2 are public input, known to Peggy and Victor

(ii) Peggy knows an isomorphism $\phi\colon V_1 \to V_2$ between G_1 and G_2, where ϕ is Peggy's *private* input – *not* known to Victor

Aim: Peggy wants to show to Victor that $G_1 \cong G_2$ without Victor learning anything about ϕ.

Use zero-knowledge proofs for this!

Proof happens in a number of rounds:

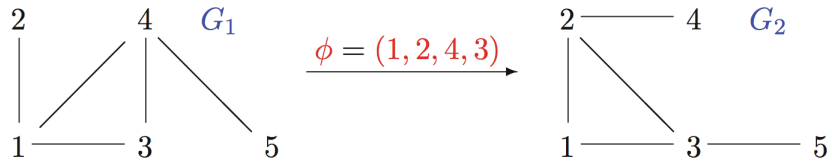
Each rounds consists of:

1. $P \to V$: H where H is Peggy's Commitment: *Peggy generates random (secret) permutation ψ on $V_2 (= V_1)$ and computes graph H from G_2 via this permutation ψ*
2. $V \to P$: b where $b \in \{1, 2\}$ is Victor's challenge: *Victor chooses b at random and sends b to Peggy*
3. $P \to V$: χ where χ: $G_b \to H$ is claimed isomorphism for $G_b \cong H$. By construction, χ has type $G_b \to H$:
   - *If $b = 1$, Peggy sets $\chi = \psi \circ \phi$*
   - *If $b = 2$, Peggy sets $\chi = \psi$.*

Now, Victor verifies whether χ is an isomorphism between G_b and H: if so, he accepts the proof; otherwise he rejects the proof.

Often, more than one round is being used: proofs are accepted only if all rounds accept!

Revisit Example:



Sanity check for us: ϕ is indeed an isomorphism (exercise).

One round of zero-knowledge proof:

1. $P \to V$: H where $H = \psi(\phi(G_1))$ and $\psi = (1, 2)$
2. $V \to P$: b:
   - $b = 1$: then $\chi = \psi \circ \phi = (1, 2) \circ (1, 2, 3, 4) = (2, 4, 3)$
   - $b = 1$: then $\chi = \psi = (1, 2)$
3. $P \to V$: χ and V can verify that χ: $G_b \to H$ is isomorphism

Note: $\chi = \psi \circ \phi$ public but ϕ secret – why is this secure?

Question: Can Peggy cheat?

Assume: Peggy has no isomorphism ϕ and so she has no witness of $G_1 \cong G_2$, moreover $G_1 \cong G_2$ may or may not be true.

In step 1: Peggy has to commit to some H based on *her* choice of some value j in $\{1, 2\}$.

In step 3:

- If $b = j$, Peggy can send the permutation she used to compute H, and Victor will verify this successfully.
- If $b \neq j$, she can only convince V if she can compute an isomorphism for $G_b \cong H$ – which is *believed to be a hard problem.* So we may assume that in this case Peggy fails to convince Victor.

Summary: Peggy has a chance of 50% of cheating in such a round.

Probabilistic amplification:

Completeness of zero-knowledge proof: Play n *independent* rounds of the above zero-knowledge protocol: graphs H and bits b are chosen at random and independently across rounds. Peggy has only probability 2^{-n} of cheating Victor in n rounds.

Soundness of zero-knowledge proof: If Peggy knows isomorphism $\phi\colon G_1 \cong G_2$, then she can convince Victor – with probability 1 – of knowing such an isomorphism in n rounds.

Recall: Victor accepts a proof iff Victor accepts *all* n rounds of that proof.

Question: In what sense are such proofs *zero knowledge*?

More formal understanding of "zero knowledge"

Simulation: show that Victor could not convince anyone that he learned something, when he presents a transcript of the n-round protocol as evidence for such a claim.

Suffices to show that Victor could simulate any such protocol run without interacting with Peggy. Since any other party would then know about this ability, Victor could not convince anybody that he learned something.

Simulation: (repeat for as many rounds as needed)

(i) Victor chooses b in $\{1, 2\}$

(ii) Victor generates random permutation χ of (vertices on) G_b to produce $H \stackrel{\mathsf{def}}{=} \chi(G_b)$

(iii) Victor generates legitimate transcript for this round:

1. $P \to V : H$
2. $V \to P : b$ same b as in step 1, possible as Victor is on control!
3. $P \to V : \chi$

Remarks

Zero-knowledge proofs often use a three-pass system of

$$\text{Commitment} \rightarrow \text{Challenge} \rightarrow \text{Response}$$

Security of zero-knowledge proofs:

$$\begin{aligned} \mathcal{V} &= \text{set of valid transcripts from } \textit{true} \text{ protocol runs} \\ \mathcal{S} &= \text{set of simulations of protocol runs} \end{aligned}$$

Degree of Security $\approx$ how similar are $\mathcal{V}$ and $\mathcal{S}$?

Perfect zero-knowledge $\stackrel{\mathsf{def}}{=}$ $\mathcal{V}$ and $\mathcal{S}$ are indistinguishable to attacker with infinite computing power.

Question: What *can* be shown with zero knowledge?

Zero knowledge and NP

NP $\stackrel{\mathsf{def}}{=}$ decision problems whose "yes" witnesses can be verified efficiently. (There is *no interaction* between prover and verifier in definition of NP.)

Examples:

(i) graph isomorphism

(ii) satisfiability of formula in propositional logic

(iii) 3-colorability of graphs

(iv) integer factorization

NP-complete decision problems $\stackrel{\mathsf{def}}{=}$ those problems in NP to which all other problems in NP reduce efficiently.

Problems (i) and (iii) above are NP-complete.

Not known whether problems (ii) and (iv) above are NP-complete, unlikely though.

CZK: complexity class based on zero knowledge

Graph isomorphism is in NP, and graph isomorphism can be proved in zero knowledge – *provided* that Peggy is computationally bounded in her computing power.

CZK $\stackrel{\mathsf{def}}{=}$ decision problems with zero knowledge for *polynomially bounded* Peggy.

Graph $G = (V, E)$ is 3-*colorable* iff $\exists \psi \colon V \to \{1, 2, 3\}$ such that $\forall (v, w)$ in $E \colon \psi(v) \neq \psi(w)$.

Theorem 21.1 3-colorability is in CZK, assuming there is a computationally concealing commitment scheme C.

Proof: $G = (V, E)$ and Peggy knows $\psi \colon V \to \{1, 2, 3\}$ as witness of 3-colorability of G. Peggy selects random permutation π of $\{1, 2, 3, \}$, and so $\pi \circ \psi \colon V \to \{1, 2, 3\}$ is another witness of 3-colorability of G.

Next, detail how Peggy interacts in a round...

Continuation of proof

1. Peggy commits to this new witness $\pi \circ \psi$ by sending Victor the commitments
$$c_i \stackrel{\mathsf{def}}{=} C(\pi(\psi(v_i)); r_i) \qquad \forall v_i \in V$$
where Peggy generates a random r_i for each v_i.

2. The verifier Victor selects a random edge (v_i, v_j) in E and sends this as challenge to Peggy.

3. Peggy sends to Victor $(\pi(\psi(v_i)); r_i)$ and $(\pi(\psi(v_j)); r_j)$ and Victor checks whether these values are really the ones that Peggy committed to in step 1:
$$\begin{aligned} c_i &\stackrel{?}{=} C(\pi(\psi(v_i)); r_i) \\ c_j &\stackrel{?}{=} C(\pi(\psi(v_j)); r_j) \end{aligned}$$
Victor accepts round if these two equations hold *and* if also $\pi(\psi(v_i)) \neq \pi(\psi(v_j))$ (so that this is consistent with G being 3-colorable).

Need to establish Completeness, Soundness, and Zero Knowledge.

Remainder of proof:

Soundness: clearly holds since Peggy will convince Victor if she knows ψ and plays as described.

Completeness: Assume that Peggy cheats. Then at least one edge (v_i, v_j) in E will satisfy $\pi(\psi(v_i)) = \pi(\psi(v_j))$. So Peggy can cheat only with probability at most $1 - \frac{1}{|E|}$.

Since graphs with 0 or 1 edge only are 3-colorable, we may assume that $|\,E\,| > 1$. So this is a proper probability.

We can repeat this proof in multiple rounds n, so Peggy can then only cheat with probability at most $(1 - \frac{1}{|E|})^n$, which converges to 0 as a sequence as $n \to \infty$.

Zero knowledge: Since $C(x; r)$ is computationally concealing, the real protocol run and its obvious simulation are computationally indistinguishable.

Facts about zero knowledge

Corollary: $NP \subseteq CZK$, assuming a computationally concealing commitment scheme exists.

Proof: Let $\mathcal{L}$ be in NP. Then there is a reduction of $\mathcal{L}$ to 3-colorability as the latter is NP-complete.

Fact: CZK is closed under such reductions.

We already showed that 3-colorability is in CZK. Therefore, $\mathcal{L}$ is also in CZK.

Therefore: *Everything in NP has zero knowledge proofs!*

- We know that $NP \subseteq PSPACE$.
- We believe that $NP \not\subseteq PSPACE$.
- We know that $PSPACE = IP$ where $IP \stackrel{\mathsf{def}}{=}$ those decision problems that can be proved with *interactive proofs* (no zero knowledge required).

So we seem to gain something through interaction!

Zero knowledge in interactive proofs
What if we want zero knowledge in interactive proofs?

Theorem 21.2: Assuming that one-way functions exist, we have that $CZK = IP$ and therefore $CZK = PSPACE$.

Definition: $f\colon \{0,1\}^* \to \{0,1\}^*$ is *one-way* iff

1. f can be computed in polynomial time and
2. for all algorithms A that run in *probabilistic polynomial time* in $n = |\,x\,|$, for all polynomials $p(n)$, and for sufficiently large n we have that

$$Pr[f(A(f(x))) = f(x)] < \frac{1}{p(n)}$$

So f is easy to compute but hard (on average) to *invert* (i.e. hard to compute a pre-image).

Fact: If there exist one-way functions, then $P \neq NP$.

It is not known whether the converse of this fact is true.

Sigma Protocols

The zero-knowledge proofs we saw so far were very inefficient:

- needed large data structures (e.g. for 3-colorability)
- probabilistic amplification rate was low (e.g. 1/2 for graph isomorphism) and so many rounds may be required to convince Victor

Honest-Verifier Zero-Knowledge Protocols: assume that the verifier responds *randomly* to commitments. In other words, challenge is uniformly distributed over all challenges and independent of the values of prior commitments.

Sigma protocol $\stackrel{\mathsf{def}}{=}$ *three-round* honest-verifier zero-knowledge protocol.

Sigma protocols have a *special-soundness property:* Victor can recover secret from two protocol runs that have same commitment but different challenge.

Special soundness implies Soundness, assuming honest verifier.

Won't prove this, but intuition is this: if Peggy does not know the secret, she is forced to change commitments across rounds, and so Victor is likely to detect any cheating.

Schorr's Identification Protocol:

- G finite Abelian group of prime order q
- g in G a generator of G, of (group) order q
- $y = g^x$ in G

Peggy wants to prove to Victor that she knows x. Here is the protocol for this efficient zero-knowledge proof:

1. $P \to V$: r where $r = g^k$ for some random k in $\mathbb{Z}_q$
2. $V \to P$: e where e in $\mathbb{Z}_q$ is randomly chosen
3. $P \to V$: s where $s = k + x \cdot e \mod q$

Victor accepts iff $r = g^s \cdot y^{-e}$.

Completeness of Schorr's Identification Protocol

Assume Peggy knows x and plays as in the above protocol. Then

$$\begin{aligned} g^s \cdot y^{-e} &= g^{k+x\cdot e} \cdot (g^x)^{-e} \\ &= g^{k+x\cdot e} \cdot g^{-x\cdot e} \\ &= g^{(k+xe)-xe} \\ &= g^k \\ &= r \end{aligned}$$

And so Victor will accept this proof.

Soundness of Schorr's Identification Protocol

As Victor is assumed to be an *honest verifier*, it suffices to establish the *special-soundness property.* Consider two runs

$$(r, e, s) \qquad\qquad (r, e', s')$$

of the protocol with the same commitment r but different challenges $e \neq e'$.

We need to show that Victor can then recover the secret x.

Note: without loss of generality Victor accepts both runs, since soundness only refers to runs that the verifier accepts. Therefore, we have

$$r = g^s \cdot y^{-e} \qquad\qquad r = g^{s'} \cdot y^{-e'}$$

Thus $g^s \cdot y^{-e} = g^{s'} \cdot y^{-e'}$ which implies $s + x \cdot (-e) = s' + x \cdot (-e') \mod q$ and so $x = (s - s') \cdot (e - e')^{-1} \mod q$.

Therefore, Victor can recover x. Since Victor is honest verifier, special soundness therefore implies Soundness.

Zero knowledge

It suffices to show that Victor can simulate any real transcript:

(i) Victor generates random $e \mod q$

(ii) Victor computes $r = g^s \cdot y^{-e}$ where s is from real transcript or generated randomly

(iii) Victor generates simulation transcript:

1. $P \to V : r$
2. $V \to P : e$
3. $P \to V : s$

The simulation transcript cannot be distinguished from a real transcript, so this shows *zero knowledge*.

Notation for Sigma Protocols

Prove knowledge of secret x.

Peggy: $R(x, k)$ computes commitment r from random nonce k

Victor: e is the challenge

Peggy: $S(e, x, k)$ computes response to challenge

Victor: $V(r, e, s)$ verifies response

Anybody: $S'(e, s)$ is simulation, creates r so that transcript (r, e, s) verifies.

Example: Schorr's Identification Protocol: $y = g^x$ in G

- $R(x, k) \stackrel{\mathsf{def}}{=} r = g^k$
- $S(e, x, k) \stackrel{\mathsf{def}}{=} s = k + e \cdot x \mod q$
- $V(r, e, s) \stackrel{\mathsf{def}}{=}$ **true** if and only if $(g^s = r \cdot y^{-e})$
- $S'(e, s) \stackrel{\mathsf{def}}{=} r = g^s \cdot y^e$

Chaum-Pederson Protocol

Has many applications, e.g. e-cash systems. Peggy wants to prove she knows x_1 and x_2 in $\mathbb{Z}$ such that

$$(1)\ y_1 = g^{x_1} \qquad (2)\ y_2 = h^{x_2} \qquad (3)\ x_1 = x_2 \mod q$$

Let us call x_1 and x_2 just x and let g and h have order q in G of order q. We specify this protocol as:

$$\begin{aligned}
R(x,k) &\stackrel{\mathsf{def}}{=} (r_1, r_2) = (g^k, h^k) && \textbf{commitment} \\
S(e,x,k) &\stackrel{\mathsf{def}}{=} s = k - e \cdot x \mod q && \textbf{response} \\
V((r_1,r_2),e,s) &\stackrel{\mathsf{def}}{=} \textbf{true} \text{ if and only if } (r_1 = g^s \cdot y_1^e \text{ and } r_2 = h^s \cdot y_2^e) && \textbf{verification} \\
S'(e,s) &\stackrel{\mathsf{def}}{=} (r_1, r_2) = (g^s \cdot y_1^e, h^s \cdot y_2^e) && \textbf{simulation}
\end{aligned}$$

Note: This protocol is similar to two runs of the Schorr protocol *in parallel*, but k is *shared* in these runs.

Completeness, Zero Knowledge, and Soundness of Chaum-Pederson

Completeness: almost the same argument as for Schorr's protocol.

Zero knowledge: just a pair of Schorr type simulations, they produce transcripts indistinguishable from real protocol transcript.

Soundness: Assume honest verifier. Therefore, it suffices to show special soundness. (Left as an exercise.)

Zero knowledge of Pederson Commitment Scheme

Recall this commitment scheme: $B_a(x) \stackrel{\mathsf{def}}{=} h^x \cdot g^a$ where both g and h are of order q with $|G| = q$.

Receiver of commitment often wants knowledge proof of x before being willing to interact more (e.g. proof that x witnesses a path on a Merkle tree in cryptocurrency).

In notation of Sigma protocols, Peggy proves $y = g_1^{x_1} \cdot g_2^{x_2}$ with g_1 and g_2 of prime order q as follows:

$$\begin{aligned}
R(x, (k_1, k_2)) &\stackrel{\mathsf{def}}{=} r = g_1^{k_1} \cdot g_2^{k_2} \\
S(e, (x_1, x_2), (k_1, k_2)) &\stackrel{\mathsf{def}}{=} (s_1, s_2) = (k_1 + e \cdot x_1 \mod q, k_2 + e \cdot x_2 \mod q) \\
V(r, e, (s_1, s_2)) &\stackrel{\mathsf{def}}{=} \text{true if and only if } g_1^{s_1} \cdot g_2^{s_2} = y^e \cdot r \\
S'(e, (s_1, s_2)) &\stackrel{\mathsf{def}}{=} r = g_1^{s_1} \cdot g_2^{s_2} \cdot y^{-e}
\end{aligned}$$

Exercise: for this Sigma protocol, show Completeness, Soundness, and Zero Knowledge.

Composition of Sigma Protocols

Sigma protocols offer not only efficiency, they also enjoy good closure properties – e.g. parallel composition, conjunction of knowledge or disjunction of knowledge.

We study such "or" proofs of *disjunctive knowledge* next.

Peggy wants to prove she knows secret x or secret y, without revealing either one of these secrets.

Assumption: We have Sigma Protocols for x and y, respectively. Challenges for both protocols come from set $\mathbb{E}$ on which we have a "one-time pad" operation $\oplus$, e.g.:

- $\mathbb{E} = \{0,1\}^*$ and $\oplus$ is exclusive-or
- $\mathbb{E} = \mathbb{F}_q$ and $\oplus$ is addition modulo q

Protocol for x: $R_1(x, k_1)$, $S_1(e_1, x, k_1)$, $V_1(r_1, e_1, s_1)$, $S'_1(e_1, s_1)$.

Protocol for y: $R_2(y, k_2)$, $S_2(e_2, y, k_2)$, $V_2(r_2, e_2, s_2)$, $S'_2(e_2, s_2)$.

Compose these protocols to obtain one for "x or y".

Zero knowledge "Or proofs"

Informal idea, given the three different possible knowledge states of Peggy:

1. Peggy knows x: Peggy runs protocol for x and simulates that for y
2. Peggy knows y: Peggy runs protocol for y and simulates that for x
3. Peggy knows x *and* y: Peggy runs either protocol 1 or 2 above.

Protocol when Peggy knows only x

- $R(x, k_1) \stackrel{\mathsf{def}}{=} (r_1, r_2)$ where $r_1 = R_1(x, k_1)$, e_2 drawn from $\mathbb{E}$, s_2 drawn from S_2, and r_2 is drawn from $S_2'(e_2, s_2)$
- $S(e, x, k_1) \stackrel{\mathsf{def}}{=} (e_1, e_2, s_1, s_2)$ where e_1 is drawn from $e \oplus e_2$ and s_1 from $S_1(e_1, x, k_1)$
- $V((r_1, r_2), e, (e_1, e_2, s_1, s_2)) \stackrel{\mathsf{def}}{=}$ **true** if and only if $e = e_1 \oplus e_2$, and $V_1(r_1, e_1, s_1)$, and $V_2(r_2, e_2, s_2)$
- $S'(e, (e_1 e_2, s_1, s_2)) \stackrel{\mathsf{def}}{=} (r_1, r_2)$ which is drawn from $(S_1'(e_1, s_1), S_2'(e_2, s_2))$.

Note that $r_1 = S_1'(e_1, s_1)$ as protocol for x is zero knowledge.

Also, Peggy won't reveal e_1, e_2 or s_2 until the response phase of the protocol.

Peggy knows only y: reverse roles of e_1 and e_2 and of r_1 and r_2 above.

Fact and Example for "Or Proof"

Fact: Let protocols for x and for y be complete, zero knowledge, and sound. Then protocol for "x or y" is complete, zero knowledge, and sound.

Example: Peggy proves knowledge of either x_1 or x_2 such that

$$y_1 = g^{x_1} \qquad\qquad y_2 = g^{x_2}$$

So Peggy wants to prove she can solve *one* of these equations for x_i.

We assume $|G| = q$ for prime q and that Peggy knows x_i but does not know x_j – in particular $x_i \neq x_j$.

Example continued

Choose $\mathbb{E} = \mathbb{F}_q^*$ and $\oplus$ as addition modulo q. Then:

- Peggy computes commitment (r_1, r_2) by selecting e_j and k_1 uniformly random from $\mathbb{F}_q^*$ and s_j uniformly at random from G. Then:

$$r_i = g^{k_i} \qquad\qquad r_j = g^{s_j} \cdot y_j^{-e_j}$$

- From the challenge e in $\mathbb{F}_q^*$, Peggy computes

$$\begin{aligned} e_i &= e - e_j \mod q \\ s_i &= k_i + e_i \cdot x_i \mod q \end{aligned}$$

Note: Replaced $\oplus$ with addition modulo q; doing so preserves relative distributions better.

Peggy outputs (e_1, e_2, s_1, s_2), and Victor checks that the equations

$$\begin{aligned} e &= e_1 + e_2 \mod q \\ r_1 &= g^{s_1} \cdot y_1^{-e_1} \\ r_2 &= g^{s_2} \cdot y_2^{-e_2} \end{aligned}$$