Introduction to cryptography

Mgr. Martin Mareš Ph.D.

February 18, 2021

Contents

1 Primitives

1.1 Basics, symmetric and asymmetric ciphers

Agreement 1.1. Alice and **Bob** are communicating parties, used only as abstraction in some protocol.
Should not be different person. E.g. disk encryption for 10 years means that the same person will decrypt in the future. Therefore Alice and Bob are the same person but time is different.

Definition 1.2. Passive attacker has an access to the communication line and listens trying to figure out text that was sent.

Definition 1.3. Active attacker can not only listen to the communication but also alter the bits.

Agreement 1.4. Passive attacker is represented by **Eve**. Active attacker is represented by **Mallory** or **Man in the middle**.

Definition 1.5. Plain text - not encrypted information that is to be delivered to from Alice to Bob.

Definition 1.6. Cipher text - encrypted plain text. Ideally properties of the Cipher text should not reveal any info about Plain text.

Definition 1.7. Encryption box (function) - converts Plain text to Cipher text. We want the box to be parametrized by **Key**.
Decryption box (function) - converts Cipher text back to plain text.

Several reasons to keep Encryption algorithm public and the Key secret (Kerckhoffs Principle):

1. Good cyphers are hard to find. As of know we know 10 strong cyphers.
2. Well known cyphers are better analyzed. We cannot proof that any cypher is strong, only hope no weakness is found.
3. Information tend to leak, there has already been cases of weak secret cyphers(cell phones protocol). Secondly, it is much easier to recover from leaked Key than cypher. For some protocols new Keys are generated periodically.

Definition 1.8. Symmetric encryption. Firstly we need Encryption function (E):

$$E : Plain = \{0,1\}^n \times Key = \{0,1\}^k \to Cipher = \{0,1\}^n$$

Denote as $E_k(x) = y$.
Decryption function (D):

$$D : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$$

Properties:

$$\forall k \forall x : D(E(x,k),k) = x \iff E = D^{-1}$$

Symmetric because Keys on the both sides are the same.

We can say that E permutes $\{0,1\}^n$. We want E to behave as a random permutation, more formally there should be no **polynomial** algorithm to distinguish between Cipher text and a Random permutation. Ciphers having that property are called Secure.

Example 1.9. Ceasars cipher - shift alphabet by some constant. Not secure at all, alphabet is that small so we can try all possible constants.

Symmetric encryption has several disadvantages:

1. For multiparties communication, new key is required for every pair. Distributing such keys secretly is not trivial.
2. Plain text size can be derived from Cipher text size.

Definition 1.10. Asymmetric cipher: E, D are parametrized by different keys. Compatible E and D should still able to decode encoded text.

$$\forall(k_e,k_d)\forall x : D(E(x,k_e),k_d) = x$$

However, deriving one key from another in pair (k_e,k_d) should be impossible for the attacker. Ideally only one k_d should correspond to a particular k_e.

Typical uses:

1. Multiparties communication. Everybody generates a key pair. k_e are public, k_d are private. Alice encrypts message with Bobs k_e, Bob decrypts it with his own k_d. Key distribution is again a problem. Alice cannot check that publicly available key, that is marked as Bobs is really created by Bob and not by the attacker. No universal solution :(.
2. Signature scheme. Alice signs a document with private k_{sign}. Everyone has access to k_{verify} to convert to Plain text. Therefore, anybody can check that message was produced by Alice. Deals with active attackers. k_e is private, k_d is public.
3. Both schemes could be combined, but we have to make sure that keys are different.

Main disadvantage if the Asymmetric ciphers is the speed of computation, in general they are slow. At least $\mathcal{O}(n^2)$. Furthermore, size of the message is bounded by the size of the Key. So, we cannot encrypt arbitrarily long messages.

1.2 Hash functions

Definition 1.11. Hash function

$$h : \{0,1\}^* \to \{0,1\}^b, b \in \mathbb{N}$$

Where b is a parameter of h, fixed # of bits.

Basic Properties 1.12.

Good hash function properties:

1. Impossible (computationally infeasible) to invert. Meaning, for given y cannot find $x : h(x) = y$.
2. Impossible to find collisions **called collision resistance**: $x \neq y : h(x) = h(y)$.

3. Random hash function. Impossible to distinguish from random function.

Example 1.13. With hash functions we can improve **signature scheme**. One can use hash function to allow multiple signatures on the same document. Alice send plain text X, then computes $E(h(X), k_{sign})$. Bob recomputes hash of the plain text and compares it with the has sent by Alice. (md5?)
Such scheme requires hash function with low probability to find collisions. Otherwise the attacker will ask Alice to sign $x : h(x) = h(y)$, then send y to Bob.

Example 1.14. Challenge-response authentication
Let's assume, Alice and Bob knows secret password X which will serve as a verification before communication starts. Alice cannot just send $h(X)$, because passive attacker could write down the password and trick Bob with it pretending to be Alice.
Bob issues challenge C first (some random text). Alice produces a concatenation of 2 strings and uses hash: $h(X|C)$.

Definition 1.15. Nonce - number used once. Long random number, e.g. Challenge in the Challenge-response authentication.

1.3 Random generator(RNG)

Unfortunately, we can only create computationally indistinguishable RNG from random function. Should be:

1. Unpredictable, even if attacker listened to the Gb of previous output.
2. Cannot be influenced (e.g. by heating the Server room)

Combining symmetric and asymmetric cipher. Firstly, Alice generate k_{sym} random, one per message. Then sends $E_{sym}(X, k_{sym}, E_{asym}(k_{sym}, k_{asym})$.
Bob takes 2nd part of the message, decrypts k_{sym} using asymmetric D. Then he decrypts first part of the message by k_{sym}.

2 Auction protocol, attacks, Security level

Observation 2.1. Any fixed bit XOR random bit (uncorrelated with fixed bit) produces random bit.

Example 2.2. Bit commitment protocol
Toss a coin over a phone call. We assume, that 2 parties communicate over some line. They want to generate a random bit, however they do not trust each other.
If Alice sends her random bit to Bob, he send his random bit back, then XOR should produce random bit. However, Bob could cheat, and choose bit according to his need.
If Alice generates a random bit, then receives Bob's random bit, then she could cheat and change her bit in the process. Similarly for Bob.
So, we need better approach. Alice generates random word of fixed length, let's say 128 bit. $x \in \{0,1\}^{128}$. Alice sends $h(x)$. Bob sends his bit b.
Then Alice send first bit of X as her random bit. Both do XOR the bits. The communication protocol ensures that Bob can verify Alice's random string by computing the hash.
X should be long, otherwise Bob could brute force to check all possible hashes.

Observation 2.3. In many communication protocols hash functions and RNG are more important than ciphers.

Example 2.4. Alice wants to participate in the auction. However, she could not do so in person. She chooses Bob as her representative.
An auction is public and it has a broadcast which cannot be changed by active attacker. Since public will notice.
Alice send 2 messages: Stop, Add <money>.
In every protocol design we should consider:

1. Against whom we are defending
2. For how long information should be secret.

In the auction example, information should be secret for a couple of minutes maximum. However, Alices actions should be kept secret before Bob acts on the auction. Otherwise other participants could do a perfect counter move.
A symmetric cipher will be used. We assume, that Alice and Bob met in person before the auction and exchanged the key.
Problems of the Protocol and the possible solutions:

1. If the messages use the same encryption key several times or, length of messages is the same, Eve could distinguish Stop from Add. Therefore, an **nonce** should be added to the message in order to make them the same length and prevent attacker from guessing using the length.
2. The difference in length between Stop and Add. To fix this, we add padding to every message making all messages the same fixed length.
3. Eve could sniff the message and check what were Bob's bid on the auction. Later she could use this message pretending to be Alice. Usually called **Replay attack**. In order to protect the communication from replay attack, Alice will add a sequence number of the current message (monotonically increasing sequence). If Bob receives the same sequence number twice, he considers it as an attack.
4. In practice, Eve could disable communication line for some time and resend all messages later. Timestamp could be a solution, however it requires a sync clock between parties. Which is hard to do secretly.

   An alternative solution would be to add NOP (no operation) message and send messages periodically.
5. Mallory can randomly change bit in the Cipher text. For many good ciphers, altering one bit in the cipher text will result in 1 bit change in plain text. To prevent that Alice should sign the message using **Message Authentication Code (MAC)**. The basic example of MAC creates $h(k_s|E_k(x))$, where k_s is a new symmetric key.

   As a result, hash of the message no longer match if Mallory changed some bits.
6. If Alice and Bob use same keys for the 2nd time, Eve could replay messages from the previous session. To prevent that they can use a different pair of keys every time. Alternatively, a **session Id** could be added to the message. Should be a secret, known only to Alice and Bob. We can also choose session id as nonce and make it a parameter of MAC.

Observation 2.5. Every part of the protocol could be attacked. Protocol is as strong as the weakest part.

Definition 2.6. Authenticity - message was not changed during transfer.
Secrecy - no one decoded the plain text of the message.
In almost all cases, we want to combine both properties.

Observation 2.7. Changing the encryption key for every message is generally no a good idea for several reasons. First of all, an attacker has much smaller set of keys to check (entropy reduction). Secondly, many RNGs require some hear up. For example, an RNG has to precompute some values.
We can however, derive key from single Master key. Later we will prove that adding **nonce** is as strong as changing the key. Frequent change of keys makes protocol harder to analyze.

2.1 Types of attacks

Example 2.8. Known cipher text - an attacker knows cipher text and wants to recover the plain text from it.

Example 2.9. Known plain text - an attacker knows cipher text and plain text. He wants to recover the Key from them. Nonce are still not known, so only part of the plain text is known.

Example 2.10. Chosen plain text - an attacker chooses plain text and gives it to the victim for encryption. Goal is again to recover the key.

Example 2.11. Distinguishing attack - e.g. distinguish Stop and Add commands in the Auction protocol (by the length or other property of the cypher text).

How to measure strength of the cryptographic primitives. Quantification. For that we define security level (in bits).

Definition 2.12. Security level - is the number of operations required to break some cryptographic primitive. Ideally we want security level to be equal to the key length. Meaning that the attacker has to check all possible keys in order to break the primitive. Security level of the good cryptographic primitive should be beyond computational power of any computer on the Earth.

Example 2.13. Birthday attack - the security level of the primitive is only half of the expected.
For example, let's consider Challenge-response auth scheme. The nonce used in the protocol has b bits. One could expect the security level to be 2^b. Which is not true, because after $2^{b/2}$ the collision is likely to occur.
This attack is related to the so called **Birthday paradox**. If there is 23 people in the room, the probability of the same birthday is actually $\geq 1/2$.

Theorem 2.14 (Birthday attack probability). *The probability of the repetitions in nonces is* $e^{-\frac{n^2}{2m}}$.

Proof. Let's compute the probability that function $f : [n] \to [m]$ that generates nonces is injective. Where n is number of runs, and m is number of nonces. Then probability of A is # of injective functions over all functions

$$Pr_f[A] = \frac{m(m-1)(m-2)...(m-n+1)}{m^n}$$

Because we choose 1 nonce every run, number of available nonces decreases by 1 every run. Then we rewrite the sum as

$$\frac{m}{m}\frac{m-1}{m}...\frac{m-n+1}{m} = 1(1-\frac{1}{m})(1-\frac{2}{m})...$$

According to Calculus, for small x: $1-x \approx e^{-x}$.

$$Pr_f[A] \approx 1e^{-\frac{1}{m}}e^{-\frac{2}{m}}... = e^{-\frac{1+2+...+n-1}{m}} = e^{-\frac{n(n-1)}{2m}} \leq e^{-\frac{n^2}{2m}}$$

For what values of n and m the probability is $\frac{1}{2}$?

$$e^{-\frac{n^2}{2m}} = 1/2 \Rightarrow \frac{n^2}{2m} = -2\ln\left(\frac{1}{2}\right) \approx 1.39 \Rightarrow n^2 \approx m$$

□

Consequence 2.15. *Number of runs required to reach $\frac{1}{2}$ probability of collisions is $\sqrt{m}$. Therefore sequrity level of Challenge-response auth scheme is $2^{b/2}$.*

Example 2.16. Assume Alice wants to send to Bob new private key, knowing his public asymmetric key k^B_{pub}. To do so, she encodes new key with Bobs key

$$E_{asym}(k_{new}, k^B_{pub})$$

Eve has access to k^B_{pub}, so she can guess a key k_g and compare the results. By the assumption, k_{new} is random, so guessing 1 key will not work. More sophisticated approach is to precompute a huge set of Keys. After some time, encrypted Key sent by Alice will be on of the precomputed one by Eve. As a result, Eve would have both encrypted and plain text of the Key.

Theorem 2.17 (Success of precomputed table). *Eve requires $tn \approx m$ (where t is size of the precomputed table, n is number of runs, m is size of Keys set) to guess the key selected by Alice. In other words, key selected by Alice will be in the precomputed subset by Eve.*

Proof. Let's compute probability that random function f generates a key that would not be in the precomputed subset.

$$P_f[f \ avoids \ subset] = (1-\frac{t}{m})^n$$

Since probability of f hitting the subset is $\frac{t}{m}$ and key selection are independent. By the same approximation as in previous theorem

$$P_f[f \ avoids \ subset] = e^{-\frac{-tn}{m}} \Rightarrow tn \approx m$$

For example $t = n = \sqrt{m}$. Again security level is $\leq 1/2$ # of bits. □

3 Perfect security, secret sharing, symmetric ciphers

3.1 Perfect Security

Definition 3.1. One time pad (also known as Vernam's Cipher) - is the only! cipher, we can prove secure.
The key $k \in \{0,1\}^n$ is chosen uniformly at random. The message $x \in \{0,1\}^n$.
cipher text is defined as

$$y = x \oplus k \iff y_i = x_i \oplus k_i$$

Decryption, using the associativity of XOR.

$$y \oplus k = (x \oplus k) \oplus k = x \oplus (k \oplus k) = x$$

Therefore encryption function is the same as decryption.
From the observation 2.1, XOR with random bits produces uniform random bit. Only the size of the information could be deduced from the cipher text. Extremely strong result! No matter how much computational power the attacker has, he cannot deduce plain text from cypher text.
For other ciphers we can only prove that brute force attack is extremely slow, or $P = NP$ should be true to break the cipher.
Such property is called **Informational-theoretic security** or **perfect security**.

Definition 3.2. Generilized one-time pad
Consider some group additive G $(G, +, 0, -)$.
We pick some $k \in G$ randomly, then

$$E(x,k) = x + k, D(y,k) = y - k = y + (k)^{-1}$$

For example we can choose $\mathbb{Z}_t$.

Definition 3.3. A symmetric cipher is *perfectly secure* iff

$$\forall x \forall y Pr_k[E(x,k) = y] = Pr_k[D(y,k) = x] = const$$

For any plain text, all cipher texts should be equally likely.

Example 3.4. For example, for generalized one time pad 3.2

$$\forall x \forall y \exists! k : x + k = y \Rightarrow k = y + x^{-1}$$

Therefore the probability is $\frac{1}{|G|}$. As a result, one time pad is perfectly secure.

Properties 3.5.

Usefulness of the OTP:

1. never use same key twice. Since xor of the cypher text is the same as xor of the plain text.

$$x_1 \oplus k = y_1 \wedge x_2 \oplus k = y_2 \Rightarrow y_1 \oplus y_2 = x_1 \oplus k \oplus x_2 \oplus k = x_2 \oplus x_1$$

2. Codebook - a long list of generated Keys. It has already happen during WW2, soviet agents ran out of keys in codebook and started to used same keys from the beginning. Not only were Americans able to read new messages, they have also decoded old ones.

3. replace randomness by pseudorandomness. A key is produced by PRNG (pseudo random number generator) which is parametrized by another key and ideally by nonce (initialization vector).

4. if attacker changes specific bit of the cipher text same plain text bit is changed. Since XOR is done bit by bit.

Theorem 3.6 (Perfect cipher). *If # of keys < # of messages ⇒ the cipher is not perfectly secure.*

Proof. Let's take some cipher text y and get all plain text that a mapped by the inverse of the encryption function to the some $T \subset$ Plain Texts. Since # of keys < # of messages $\Rightarrow$ some of the keys $\notin T$. Therefore

$$Pr_k[x \notin T \wedge D(y,k) = x] = 0, Pr_k[x \in T \wedge D(y,k) = x] > 0$$

And the distribution of keys are not uniform. Cipher is not perfectly secure by the definition. □

Note 3.7. Perfect security is also called **Shannon security**. He is the father of the information theory and cryptography.

3.2 Secret Sharing

Assume we have a secret information. We want to break secret into shared. The whole secret can be restored only having all the pieces. We can use the OTP:
$S_1, ..., S_{k-1}$ are random numbers. $S_k = x \oplus S_1 \oplus ... \oplus S_{k-1}$. Shareholders can decode x by xoring all S_i. Otherwise they will get random numbers.

Definition 3.8. (k,l) - threshold scheme. Split secret x into k shares $S_1, ..., S_k$. Having any l shared agents can decode x. However X cannot be decoded with $< l$ # of shares. No info should be given.

Example 3.9. $(k,2)$ scheme. Let F be finite field, the secret is a function $f = ax + b$ s.t.:

$$f(0) = x, f(1) \in_R F$$

Geometrically, we have x as the point with coordinates $(0,t)$ for some t. All S_i for a straight line. As line can be deduced by any 2 points, 2 agents can decode x using their points S_i.
Distribution of the secrets x is uniform ⇒ perfectly secure.

Let's consider polynomials over field F. According to results from Algebra, a polynomial of degree d has at most d roots.
Graph of the polynomial is a set of values for every point $\in F$. Polynomials with the same graph share the roots ⇒ should be equal.

Theorem 3.10 (Lagrange interpolation). *For every $x_1, ..., x_d, x_i \neq x_j, i \neq j, P(i) = 0$ distinct roots of P and $y_1, ..., y_d$*

$$\exists! P, deg(P) < d : \forall i \ P(x_i) = y_i$$

Proof. Let $\overline{x} = (x_1, ..., x_d) \in F^d$ be a vector of coefficients of polynomial P, X is a set of all such vectors. $|X| = |F|^d$.
Let Y be a set of all choices of y_i, graphs of the polynomials.
Finally, let $f : X \to Y$ be a map from polynomials to graphs of polynomials. Which evaluates polynomial at d points.
Since polynomials with the same graph are equal, f is injective. $|X| = |Y| \Rightarrow f$ is bijection. □

Consequence 3.11. *Scheme based on graphs of the polynomials is perfectly secure, since according to the Lagrange theorem we have bijection between them ⇒ uniform distribution.*

Example 3.12. (k, l) scheme. Instead of lines we pick some polynomial P, $deg(P) = d$. Then we randomly pick k values of the P. Having exactly k shared agents can obtain P. Otherwise, all $X = P(0)$ are uniformly distributed.

4 Symmetric ciphers, DES, AES

4.1 Symmetric Ciphers

There are 2 kinds of symmetric ciphers: block and stream.

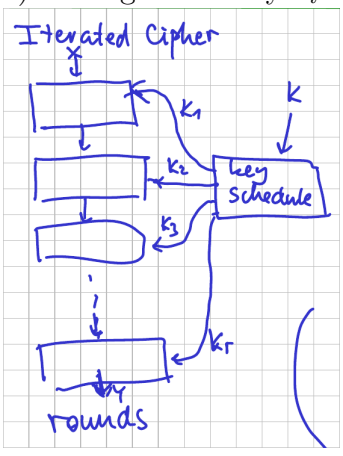
Definition 4.1. Block ciphers work on fixed-size blocks of b bits.

$$E : \{0,1\}^b \times \{0,1\}^k \to \{0,1\}^b$$

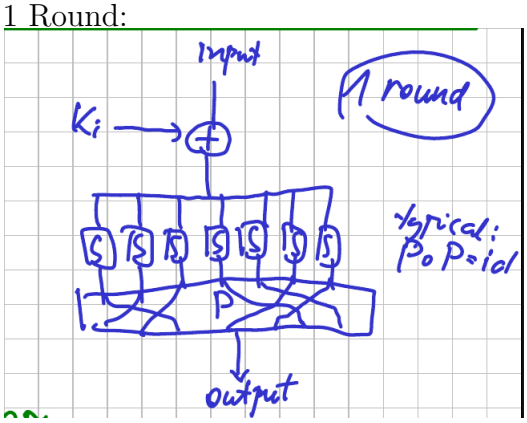
Therefore, encryption is a bijection (permutation on set of block values).

Definition 4.2. Assume an attacker is given an oracle which contains either E_k with random key k or a random permutation. Block cipher is secure iff an attacker cannot distinguish what is inside the oracle.
More precisely: does not exists a distinguisher with $Pr[\text{success}] \geq 2/3$ and run time $< 2^{security\ level}$.

Definition 4.3. Substitution-Permutation Network (SPN) is an iterated cipher where each round consists of:
1) S-boxes of bijective substitution (confusion)
2) P-box (permutation) on positions (diffusion). Typically permutation is an involution: $\pi \circ \pi = id$.
3) Mixing round key k_i by $\oplus$.



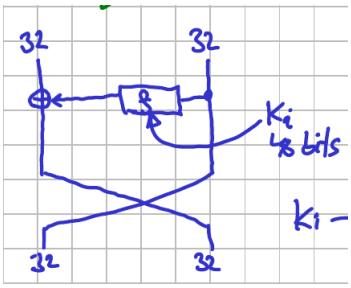
1 Round:



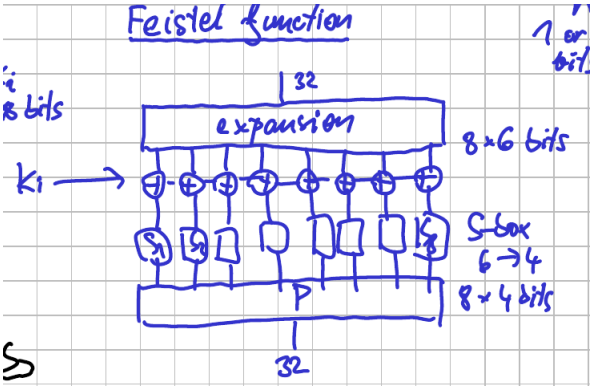
Round is invertible. Inverse of SPN is again an SPN.

4.2 DES

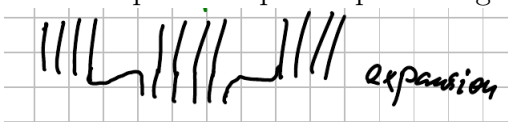
DES contains Feistel network with 16 round, 64-bit blocks, 56-bit keys.



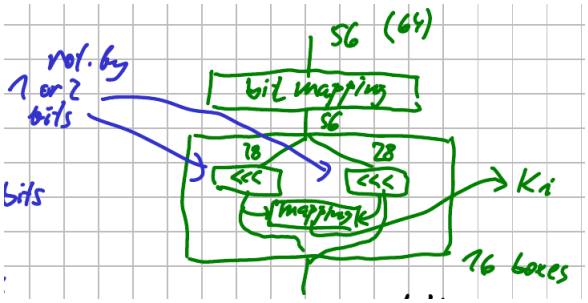
Feistel network (1 round):
Where f is non-invertible function called Feistel function. K_i is a new key derived from master, one per round.



Feistel function:
Where expansion splits input into groups of 4 and takes 2 bits from neighbours.



Key schedule Produces new keys per round:



Bit mapping discards parity bit and permutes the rest. As a result key is 56b, instead of 64b as formal definition. Key schedule produces a new key, also sends output to the new round (of key schedule).
2nd mapping box is the same for all round in Key schedule, the only difference is in the # of rotations in << (1 or 2 bits rotation).
As a result every K_i is a subset of master key, permuted in some way.
Critique of DES:

1. Weak keys. If $K = 0^{56} \Rightarrow \forall i : K_i = 0^{48}$, so $\oplus$ after Feistel function has no effect and all rounds are identical. Same for all 1s.
2. $E_{\overline{k}}(\overline{X}) = \overline{E(x)}$. Random symmetric cipher does not have this property.
3. Short key. Brute force attack is feasible (26 hours in 2012). Solution: use 2-DES or 3-DES.

   2-DES contains 2 round of DES. Does not improves security level significantly, only to 2^{57}. With that many steps known plain and cipher text we can break the key.

   3-DES: $E_{k_1}(D_{k_2}(E_{k_3}(x)))$. Which is slow, security level is ≤ 113, 168 bits of key.
4. Too short blocks: in 2^{32} block collision will occur. Solution: change the key after $\approx$ mln of blocks.
5. Too much secrets (NSA changed s-boxes last minute)
6. Attacks on structure (2^{47} chosen plain text)

To conclude, DES is dead!.

4.3 AES

Definition 4.4. AES - advanced encoding standard, known as Rinjdael. Has 128 bit blocks, multiple key lengths in standard:
1) 128b (10 rounds)
2) 192b (12 rounds)
3) 256b (14 rounds)
Internal structure: SPN with linear step.
Internal state: matrix $M \in GF(2^8)^{4\times 4}$.

AES round:

1. (S) Byte sub (16 identical s-boxes). Does inversion in GF + affine transformation.
2. (P) Shift rows. //TODO picture
3. (L) Mix columns: liner transformation on every column

4. $\oplus$ XOR with K_i.

Decryption does inverse:

1. $\oplus$ XOR with K_i.
2. (L) Inverse Mix columns (additional linear step): liner transformation on every column
3. (P) Inverse Shift rows.
4. (S) Inverse Byte sub (s-box should be invertible).

Observation 4.5. (S) and (P) commute, order can be changed. We can also swap XOR and Inverse mix columns with a key modification.
Consequently, D and E differ only in cleanup.

Note 4.6. SW implementation of AES could be speeded up by precomputing Byte sub as lookup table. Also precompute

$$\begin{aligned} m_1(x) &= mix(s(x), 0, 0, 0) \\ m_2(x) &= mix(0, s(x), 0, 0) \\ m_3(x) &= mix(0, 0, s(x), 0) \\ m_4(x) &= mix(0, 0, 0, s(x)) \end{aligned}$$

Then mix is

$$mix(s(x_1), s(x_2), s(x_3), s(x_4)) = m_1(x_1) \oplus m_2(x_2) \oplus m_3(x_3) \oplus m_4(x_4)$$

Requires only 1kb of precomputed values. However can be attacked because of cache properties.

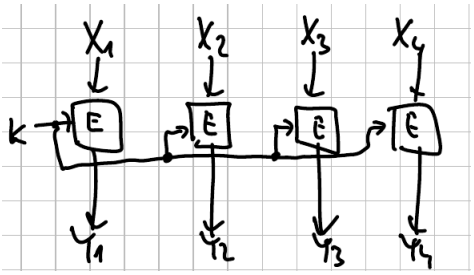
Critique of AES:

1. Simple algebraic structure (LA + finite fields) $\Rightarrow$ could lead to system of Linear equations that could break the key.
2. Small margin in # of rounds. (Many attacks start with less rounds e.g. 6 and succeed, than increase number of rounds)
3. Byte alignment. All operation are on whole bytes $\Rightarrow$ diffusion and confusion are much faster across bytes than inside.
4. 128-bit key can be attacked by quantum computers using Grover alg. Which requires $\sqrt{n}$ steps comparing to n steps of brute force.
5. 128-bit blocks $\Rightarrow$ collisions in $\approx 2^{64}$ blocks. Solution: change key after 2^{32} blocks.

4.4 Usage and Modes of Block ciphers

Because of the block cipher properties we need:
1) Padding that can be reversed.
2) Split message into blocks

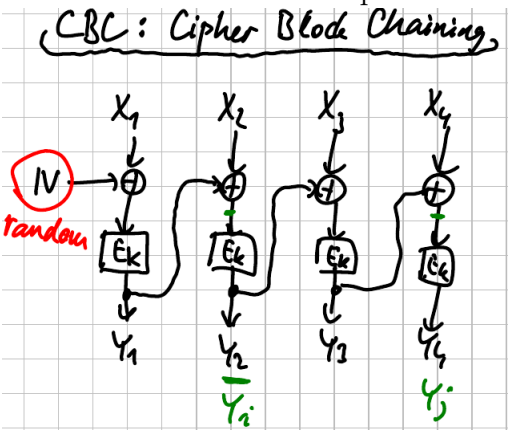
Definition 4.7. ECB - electronic code book. Encrypts all blocks independently in parallel.



Avoid at all costs:

1. No IV.
2. Encryption is the same for identical inputs.
3. Reveals $x_i = x_j \Rightarrow y_i = y_j$ blocks.
4. Attacker can change 1 bit in $y_i \Rightarrow$ destroys x_i.
5. Swap $y_i = y_j \Rightarrow$ swaps $x_i = x_j$.

Definition 4.8. CBC - cipher block chaining.



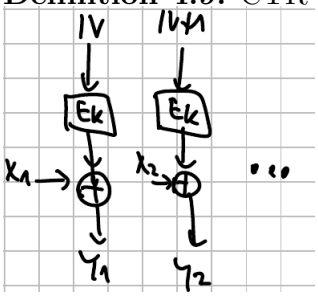
Properties:

1. Requires random IV.
2. Bit flip in $y_i \Rightarrow$ destroys x_i and flip in x_{i+1}.
3. Block swap gives predictable result:

$$y_i \iff y_j \Rightarrow x_{i+1} = y_i \oplus y_j \wedge x_i = x_j \oplus y_{i-1} \oplus y_{j-1}$$

4. Proof of security for chosen plain text. CBC is secure even using not ideal cipher for short messages $\leq (\sqrt{block\ space\ size} \cdot numb\ blocks)$.

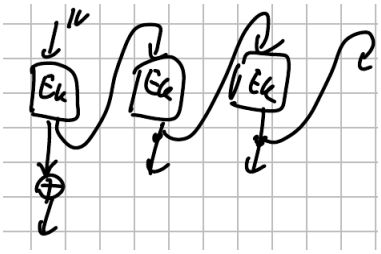
Definition 4.9. CTR - counter.



Properties:

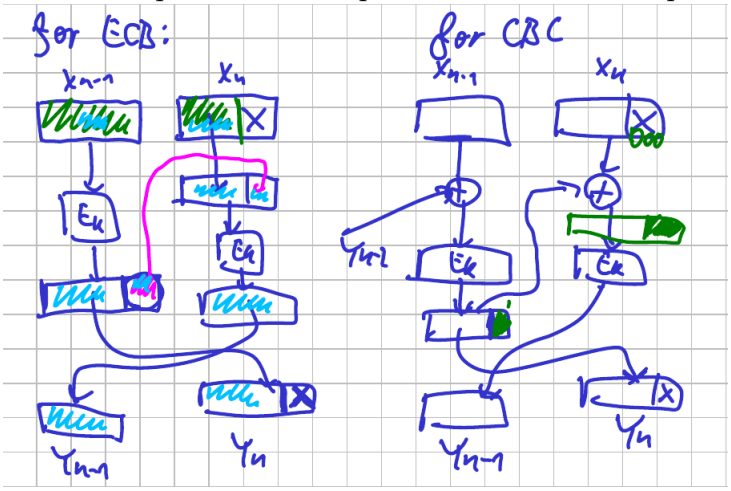
1. Behaves as stream cipher $\Rightarrow$ no need of padding.
2. Cannot use same IV twice.
3. Bit flit in $y_i \Rightarrow$ bit flip in x_i. Should be handled by protocol.
4. Random access + parallelizable $\Rightarrow$ suitable for disk encryption.

Definition 4.10. OFB - output feedback.



A short cycle of permutations could occur $\Rightarrow$ reduces # of IVs as we cycle.

Definition 4.11. Cipher text stealing - trick to avoid using predictable padding pattern for block ciphers. We use plain text data to complete last block.



All block ciphers leak some information, which is significant.

1. ECB: $x_i = x_j \Rightarrow y_i = y_j$.
2. CBC: $y_i = y_j$ is likely to happen after $2^{b/2}$ blocks. Also

   $$E_k(x_i \oplus y_{i-1}) = E_k(x_j \oplus y_{j-1}) \Rightarrow x_i \oplus y_{i-1} = x_j \oplus y_{j-1} \Rightarrow x_i \oplus x_j = y_{j-1} \oplus y_{i-1}$$

   b bits are known to attacker per $\approx 2^{b/2}$ blocks. Solution: do not use a single key for encryption of $2^{b/2}$ blocks.
3. CTR: Assume $C_1, ..., C_m : C_i = E_f(IV + i - 1)$. All C_i are different, so

   $$y_i \oplus y_j = (x_i \oplus C_i) \oplus (x_j \oplus C_j) = (x_i \oplus x_j) \oplus (C_i \oplus C_j) \wedge C_i \oplus C_j \neq 0 y_i \oplus y_j \neq x_i \oplus x_j$$

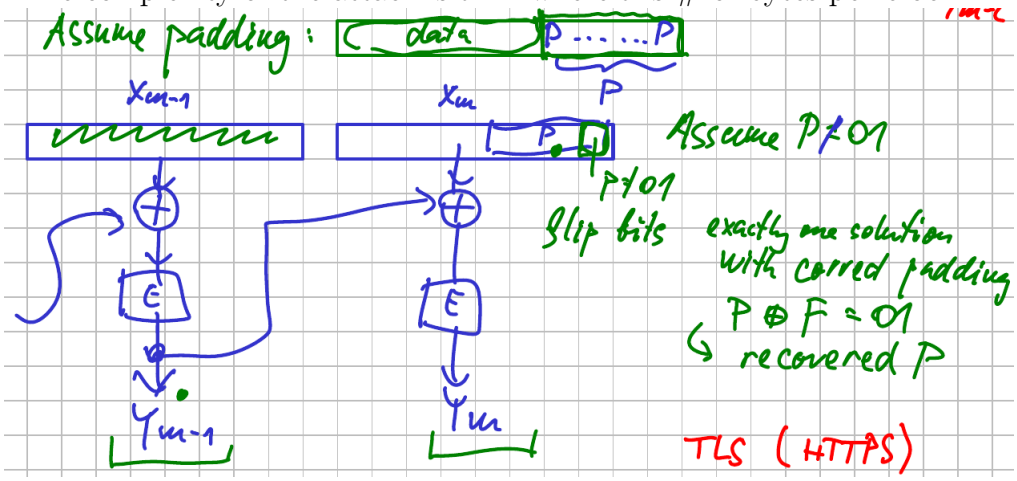
   Rules out 1 of 2^b possibilities for every pair. And the difference in entropy per pair is:

   $$b - log(2^b - 1) = \log\left(\frac{2^b}{2^b - 1}\right) = \log\left(1 + \frac{1}{2^b - 1}\right) \approx c2^{-b}$$

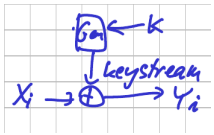
   So # of bits leaked $\leq \binom{m}{2} \cdot c \cdot 2^{-b}$, const for $m \approx 2^{b/2}$.

4.5 Padding oracle attack

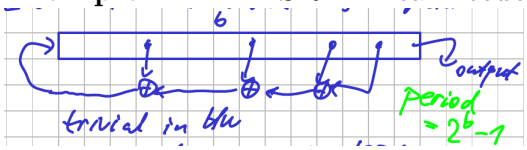
Assume, we use an padding pattern with p block filled with value p in every byte. Attacker has access to an oracle (e.g. server) that for given cipher text answers whether padding was correct, or data is corrupted. For example by a return code (500), alternatively attacker can measure how long it took to process the data (time attack).
Because of the CBC mode structure, changing y_{n-1} block (which is XORed with decrypted y_n) changes last block of plain text. Assume $p \neq 1 \Rightarrow$ last byte is not 01. An attacker can give server all possible values of the last byte to get 01 in the decrypted plain text. If we guessed such value, we get last byte of decrypted y_m (intermediate state). The last step is to XOR initial y_{n-1} with intermediate state we have. Doing so we obtain last byte of plain text, which is not only equal to padding, but also reveals how many block of padding plain text had.
Attack will continue by changing cipher text to get padding equal to $(p+1)$ till we reach maximum possible padding. Then attacker just removes the last block of cipher text and repeats first phase of attack.
Time complexity of the attack is $b \cdot 2^8$ where b is # of bytes per block.



4.6 Stream ciphers



Example 4.12. LFSR - Linear Feedback Shift Register.

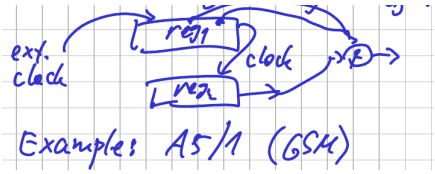


Implementation is trivial in HW (bit shifts + XOR). However, LFSR can be cracked by Known plain text attack.

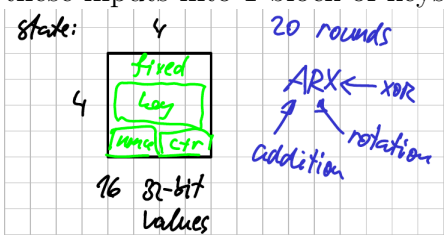
Attempts to save LFSR:

- non-linear feedback (&)
- non-linear output (e.g. using hash function).

- combine multiple registers and hash together.
- control clock of LFSR using second register.



Example 4.13. ChaCha20 is a stream cipher developed by Bernstein. Consists of 20 rounds. Has 256b key, 64b nonce, 64b block counter. A non-bijective function f converts these inputs into 1 block of keystream.



Salsa20 is the upgraded version of ChaCha20.

5 Hash functions, Merkle trees, MACs

Definition 5.1. Hash function f is

$$f : \{0,1\}^* \to \{0,1\}^b$$

Typically used for signatures.

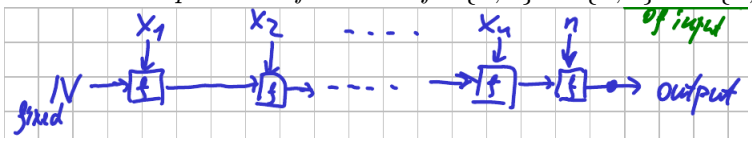
Properties 5.2.

Secure hash function:

1. collision-free: there are no $x \neq y : h(x) = h(y)$
2. no second preimage: for given x, find y: $h(x) = h(y)$.
3. no inversion: for given z find $x : h(x) = z$.

5.1 Merkle-Damgard construction

Given a *compression function* $f : \{0,1\}^b \times \{0,1\}^b \to \{0,1\}^b$. Construct h:



Theorem 5.3 (Collision resistant hash). *If f is collision resistant compression function $\Rightarrow$ h is collision resistant hash function.*

Proof. Indirect proof. Assume we found a collision of h, meaning $h(x_1, ..., x_n) = h(x'_1, ..., x'_n)$
Let's consider 2 options:
1) $n \neq n'$. In the last step we compress n and n'. As outputs are identical $\Rightarrow$ the last

block containing size is also identical. Therefore we have found collision of f.
2) $n = n'$. Using induction we examine compressed blocks. As outputs are identical: either the blocks were identical or we have found collision of f. Then, as messages are not identical, there exist two different blocks encoded to the same output. Which implies collision of f. □

5.1.1 Length extension property of M-D hash

Suppose we have a message x and prefix p. For a Random function, knowing prefix $h(p)$ should help us producing full $h(p|s)$. Where $s = x - p$ suffix.
From the M-D internal structure, hash of the suffix depends on the hash of prefix. Therefore, an attacker, knowing hash of the prefix, can add his data to the message and compute new signature (hash).
We should take this property into consideration using M-D hashes.

5.1.2 Compression function for M-D

Having a block cipher, we can use it to create compression function as the following:

$$f(u, v) := E_u(v) \oplus v$$

XOR is required because otherwise finding collision is easy. Suppose $f(u, v) := E_u(v)$, then

$$E_u(v) = y$$

Taking another key, e.g. b, we decrypt y using b

$$D_b(y) = a \Rightarrow f(u, v) = y = f(a, b)$$

Using DES as block cipher is also a bad idea, since $E_{\overline{k}}(\overline{x}) = \overline{E_k(x)}$:

$$f(\overline{a}, \overline{c}) = E_{\overline{a}}(\overline{c}) = \overline{E_a(c)} \oplus \overline{c} = E_a(c) \oplus c = f(a, c)$$

For $v := D_u(0), f(u, v) = E_u(v) \oplus v = 0 \oplus v = v$. Which cannot happen for random hash function.

Theorem 5.4 (Collision resistant hash). *With an ideal block cipher (for every key a random permutation) f is collision resistant. In particular, for an attack evaluating E/D q times ($q \leq 2^{b/2}$)*

$$Pr[collision] \leq q^2/2^b$$

Proof. WLOG: attacker do not ask redundant questions (E of the text he already knows). From the properties of ideal block cipher, results of encryption is a bijection. Can be viewed as table of values.
Asking E and D questions attacker gets following answer:

$$E_u(v) = f(u, v) = E_u(v) \oplus v \wedge D_u(v) = f(u, D_u(v)) = v \oplus D_u(v)$$

We find first collision in step $i : 1 \geq i \leq q$. We found $f(l, k)$ matching some known value, having $(i - 1)$ of them. So it was either the answer to E or D question, which is exactly 1 value.

For every pair, known and new answers:

$$Pr[collision] = \frac{1}{num\ of\ possible\ ans.} \leq \frac{1}{2^b - i - 1} \wedge i \leq 2^{i/2} \leq 2^{b/2} \Rightarrow Pr[collision] \leq \frac{1}{2^{b-1}}$$

$$Pr[collision] \leq \frac{1}{2^{b-1}} pairs \leq \frac{1}{2^{b-1}} \binom{q}{2} \leq \frac{q^2}{2 * 2^{b-1}} = \frac{q^2}{2^b}$$

□

5.1.3 Finding collision of hash function

1. Brute force: by the Birthday paradox expect a match in $\approx 2^{b/2}$ steps. Lots of memory.

2. With constant memory. Assume $\forall x : |x| = |h(x)|$. Let's construct a graph where vertexes are all possible blocks and edges are $(x, h(x))$. Degree of every x is 1. Iterating $h(h(...h(x))$ we create a path in Graph. At some point there will be a loop, as Grah is finite.

   A subgraph reachable from x looks like lollipop. Path + cycle. In order to find collision in such subgraph we choose 2 travelers in the graph: tortoise (1 step at time) and rabbit (2 steps at time). Starting at x, they move in graph. Claim: they will meet in vertex which connects path with cycle.

3. In many cases we need to find meaningful collision (to send evil message instead of innocent). And the party signing messages refuses to sign random noise.

   In order to find collision, we construct parametrized message. Let's say Dear Bob, Dear Sean etc. With k places of parametrization we can produce 2^k innocent messages.

   Both previous attacks work with such messages. However, edges in the Graph for 2nd type of attach will be $(x, h(parametrize(x)))$.

4. Meet-in-the-middle attack. We want $h(evil) = h(innocent)$.

   Let A,B two random subsets of all hashes. Than, with high probability they have non empty intersection. $|A| = |B| = \sqrt{n}$. A will be hashes of $2^{b/2}$ innocent messages, B - hashes of $2^{b/2}$ evil messages.

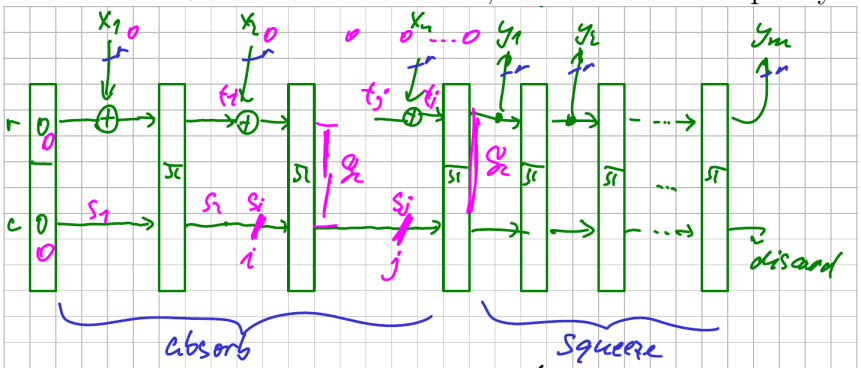
5. For M-D hashes we can product lots of collision, as easy as 1. By Birthday paradox, we find first collision: $x_1 \neq x_1' : f(IV, x_1) = f(IV, x_1') = y_1$. Then we repeat k times using y_{i-1} as IV. As we can use both x_i and x_i' to get y_{i-1}. We get 2^k combinations which hash to the same result.

   As a result, concat of 2 hashes, both with length b does not have security level $2b$ if either of functions is M-D. Because we can find $2^{b/2}$ colliding messages in time $b/2 * 2^{b/2}$. Than h_2 is likely to have 2 collisions. Therefore, we have found collision in $h = h_1|h_2$ in time $b/2 * 2^{b/2}$.

5.2 Sponge construction

Sponge construction has 2 phases:
1) Absorbing the input (updating internal state depending on input)
2) Squeezing the output

Example 5.5. SHA-3 sponge construction. Suppose we have a permutation π on blocks of size $w = r + c$. Where w is width, r - rate and c - capacity.



Security against BF attacks:
1) By the Birthday paradox we can attack the output in 2^r steps.
2) Internal collisions (chosen plain text attack):
By the Birthday paradox in 2^c steps we can find

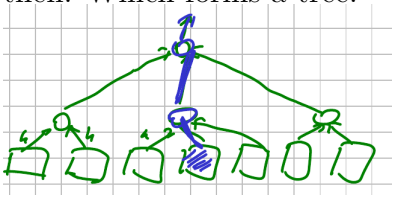
$$i < j : S_i = S_j$$

In order to get collision, we feed the sponge 0^i as first part of input. Then we feed $(t_i \oplus t_j)$ to get $r = t_i$ back and feed zeros till the end 0^{j-1}. As a result, after j-th stage sponge squeezes the same output.
However, we know that for a random π security level $\geq \min(r/2, c/2)$.

5.3 Other hash functions

Shake-128, an XOR(extensible-output functions) Designed for variable size output. Can be used as PRNG, that is initialized by some value. Can be run for many # of steps.

Definition 5.6. Merkle trees. Divide data into blocks, then hash and concat some of then. Which forms a tree.

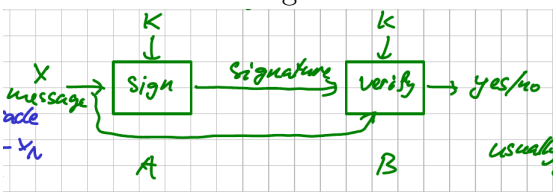


Claim: if hash function f is collision-free $\Rightarrow$ whole tree is collision-free.
Can be run in parallel, also updating a single block of data result in path update only.
Used in practice to sign a DB state.

Observation 5.7. Naive Merkle tree is not secure to *extension attacks*. As we can take old tree, add another subtree(that includes our evil data) and a new root.
Fix: add 1 bit to every node, 1 is for Root. (see Sakura coding as example).

5.4 MACs

MACs are used to sign some information (either plain or cipher text).



Usually, sign is deterministic and verification is just sign + compare with received signature from the counterparty. Typically sign is parametrized hash.

Security Model Attacker has access to *signing oracle* and can ask it to sign $(x_1, ..., x_n)$ plain texts. Cryptographic primitive is secure iff the attacker is able to produce new valid signature for $x \neq x_i$.
How to sign? One can include a key to the hash alongside the text. There are several ways to do so:

- $h(k|x)$ is secure for random h, but not for Merkle-Damgard. M-D hashes are not protected against extension attacks. SHA-3 is believed to be secure for such usage. Such construction is included to the standard as **KMAC**.
- $h(x|k)$ will not work either, because the attacker can precompute all combinations independent from key, then try with specific key. Therefore, $h(x|k)$ should be also avoided.
- Much better option is $\mathbf{HMAC}_h := h(k \oplus C_{out}|h((k \oplus C_{in})|x))$. C_{in}, C_{out} are derived from master key. Where $h((k \oplus C_{in})|x)$ is collision-resistant. Consequently, we can even use broken SHA-1,2.

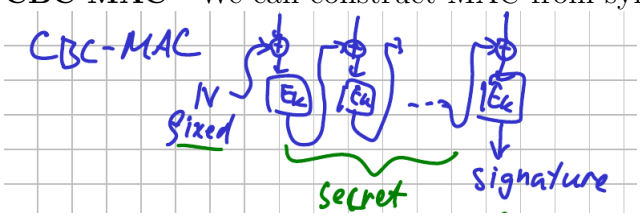
Combining auth with enc Typically, we want both security and authenticity, so that the attacker cannot neither decrypt nor change the message. To achieve that, MAC with encryption is used. Again, there are several possibilities to combine 2 primitives.

- Encrypt and MAC independently is a bad idea, since the signature of the same message encrypted with different cipher is leaking their identity.
- MAC then Encrypt has several advantages over the opposite option:
  1) Signature should reflect plain text as close as possible. Should be same for different ciphers.
  2) Authenticity is more important than secrecy (e.g. for nuclear missiles control system)
  3) MAC is protected by the cipher

  Unfortunately, described scheme is vulnerable to the *oracle timing attack*. Specifically, reducing the padding by 1 block will result in 1 MAC iteration less. Which can be observed by the attacker. As a result, almost all protocols that includes MAC then Encrypt are broken because of this attack.

  However, MAC then Encrypt is used in TLS, To avoid the attack, chosen MAC is not the block type.
- Encrypt then MAC is the recommended scheme.

CBC-MAC We can construct MAC from symmetric cipher in CBC mode.



Properties 5.8.

In order to be secure, CBC-MAC should have these properties

- Fixed IV. If the IV is random (not fixed), it should be transfered over the network. As a result, Mallory can flip bits in both IV and the first block of the message. Which will alter the message, but the sign remains valid.
- Intermediate steps of the CBC should be kept secret.
- No message is a prefix of another message.

With these properties, assuming ideal cipher, CBC-MAC is proved to be secure.

Definition 5.9. An MAC is *Shannon secure* iff given a known pair $(x, sign(x))$ all signatures of $x' \neq x$ are equally possible for a random key. Where $sign(x)$ is a signing function which uses a single random key.
Alternatively, a pair $(x, sign(x))$ gives no info about other messages.

Definition 5.10. A family of functions from set X to set Y is $\mathcal{H} := \{h_k | k \in \mathcal{K}\}$. Technically a multiset.

5.4.1 2 independent MAC

Definition 5.11. $\mathcal{H}$ is 2-independent iff

$$\forall x, x' \in X \forall y, y' \in Y : Pr_h[h(x) = y \wedge h(x') = y'] = \frac{1}{|Y|^2}$$

Typically we use Galois field (GF) as X and Y.

Example 5.12. $h_{a,b}(x) := ax + b$ is a 2-independent family of functions. To construct a MAC from this family we:
1) select $h \in \mathcal{H}$ at random
2) key is (a, b) - parameters of h
3) signature is $h(x)$.
To prove 2-independence we calculate (x_1, x_2, y_1, y_2 are given constants):

$$Pr_f[h_k(x_1) = y_1 \wedge h_k(x_2) = y_2] = Pr_{a,b}[ax_1 + b = y_1 \wedge ax_2 + b = y_2]$$

As we have system of 2 linear equations with 2 unknowns: a and b, it has an unique solution. Therefore the probability is $\frac{1}{|Y|^2}$, all possible choices of parameters a, b.

Theorem 5.13 (2-ind MAC security). *2-ind MAC is Shannon secure.*

Proof. By the conditional probability:

$$Pr_f[h_k(x_1) = y_1 | h_k(x_2) = y_2] = \frac{Pr_f[h_k(x_1) = y_1 \wedge h_k(x_2) = y_2]}{Pr_f[h(x_1) = y_1]}$$

By the 2-independence

$$Pr_f[h_k(x_1) = y_1 \wedge h_k(x_2) = y_2] = \frac{1}{|Y|^2}$$

$Pr_f[h(x_1) = y_1]$ can be calculated as a disjoint sum of events

$$\sum_{y_i} Pr_h[h(x) = y \wedge h(x_i) = y_i] = \frac{|Y|}{|Y|^2}$$

Combining 2 results

$$Pr_f[h_k(x_1) = y_1 | h_k(x_2) = y_2] = \frac{1}{|Y|^2} \div \frac{1}{|Y|} = \frac{1}{|Y|}$$

Which satisfies Shannon security. □

Downsides of 2-ind MAC:

- key is twice as large as a message.
- no key can be reused
- sequence number is required
- message size = signature size

5.4.2 Polynomial based MAC

Suppose, F is Galois field, $X = F^n, Y = F, K = F^2$. Then

$$h_{a,b}(x_1, ..., x_n) = x_1 a^n + ... + x_n a + b$$

Evaluation the polynomial at some point could be done in $\mathcal{O}(n)$.

Theorem 5.14 (Polynomial MAC security). *Polynomial MAC is not Shannon secure.*

Proof.

$$Pr_{a,b}[h_{a,b}(x'_1, ..., x'_n) = y' | h_{a,b}(x_1, ..., x_n) = y] = Pr_{a,b}[L|R] = \frac{Pr[L \wedge R]}{Pr[R]}$$

Substracting 1st equation from second we get

$$Pr[L \wedge R] = (x_1 - x'_1)a^n + ... + (x_n - x'_n)a = y - y'$$

Which is a non zero polynomial, since messages are different $\Rightarrow$ at most n values of a. Also, initial equation was satisfied for exactly one $b \Rightarrow$ at most n pair (a, b). Yielding:

$$Pr[L \wedge R] \leq \frac{n}{|F|^2}$$

Then

$$Pr[R] = \frac{|F|}{|F|^2} = \frac{1}{|F|}$$

And

$$Pr_{a,b}[L|R] \leq \frac{n}{|F|}$$

Which does not satisfy Shannon security, as n is in numerator. However, we can choose reasonably big Field.

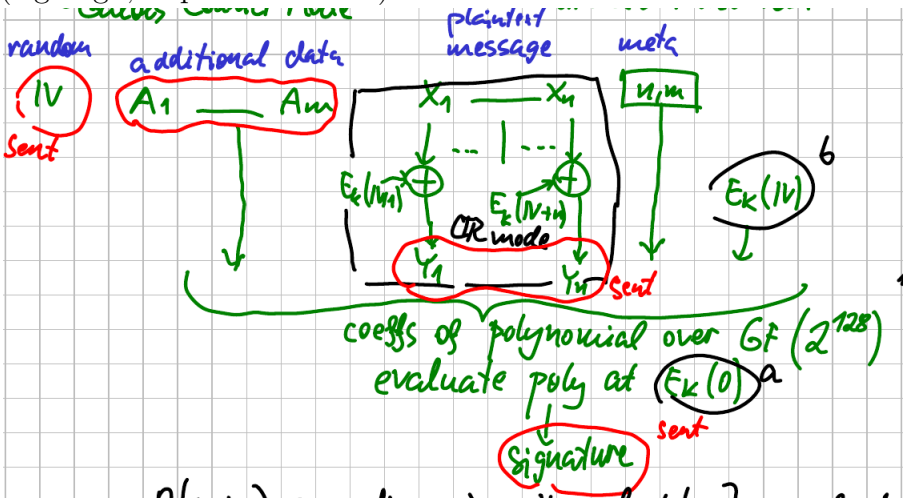
□

And yet, polynomial MAC is not practical as we need a new random key for every message. Which can be solved by using PRNG parametrized by key.

Observation 5.15. If parameter a is fixed and b is generated by PRNG security level does not decrease.

5.4.3 GCM mode of block ciphers

Definition 5.16. GCM (Galois counter mode) is an auth encoding with additional data (e.g. sign, sequence number)



We use additional data, encoded data, sizes and IV as a coef of a Polynomial. And $P(E_k(0))$ is a signature, where $E_k(0)$ is a constant than has to be secret!
As elements are 128-bit strings, addition is XOR multiplication of polynomials over field $\mathbb{Z}_2$. Which is a *carry-less* multiplication mod some irreducible polynom Q.

Example 5.17. Poly 1305 [Bernstein], is a MAC construction from block ciphers over $GF(2^{130}-5)$. Size of the field is choosen close to power of 2 for speedup. Moreover, points a in which polynomial is evaluated are also some specific subset of the field for the same reason. Security level does not decrease significantly.

6 Random number generators

Definition 6.1. Secure RNG - even knowing complete past output, next bit is unpredictable $\Rightarrow$ statistically uniform.

We have 2 type of random number generators:
a) PRNG: e.g. block cipher in CTR mode. Key should be changed on regular basis.
b) physical randomness (we have no proof such thing exists). Examples:

- Thermal noise or resister/diode
- Radioactive decay (count # of particles)
- Radio noise
- Lava lamp (chaotic movement)
- Ring oscillator (attacker can observer)
- Precise timings of keystrokes, mouse movements or network packages (attacker can record or flood packages)

Note 6.2. Entropy in physical processes is not as high as we can expect. Therefore, a combination of PRNG and physical randomness is the best choice.

Problem with PRGN + rekey: if a key or state of the PRNG is compromised, adding 1 bit or physical entropy does not help. Attacher can try both options (guess) and track the state. Therefore entropy should be mixed in large batches.

6.1 Fortuna RNG

Definition 6.3. Fortuna RGN has a *Generator* and *Accumulator*.
Generator is based on block cipher (AES) which encrypts a 128-bit counter (never overflows). After 2^{16} block we rekey and using the next 2 blocks as key.
Trick: do not reset the counter to avoid short cycles.
Accumulator: gets entropy from other sources and resets Generator periodically. Has $p_0, ..., p_{31}$ pods of entropy. External RNG is mixed to the pods in Round-Robin order: assign RNG to p_i, move to the next. Such process distributes RNG among the pods.
After 100ms we mix pods into Generator. In the j-th mixing only selected pods are used, e.g. with index $2^j/i$.

Assume that $\rho =$ rate of input entropy, measured in bits per 100ms. If $\rho \geq 128 * 32 \Rightarrow p_0$ is enough for recovery. As at least 128 bit of entropy were put in p_0. If $\rho \geq 128 * 32/2^i \Rightarrow p_i$ is enough to recover.
To conclude, for any given rate of entropy Fortuna can recover after some time. Speed of recovery depends on entropy rate.

6.2 Secure channel

We use symmetric encryption to establish secure channel. Assume Alice and Bob share unique random key k. Alice sends $\{m_i\}_0^n$ sequence of messages, Bob receives a subsequence and he is still able to know which one he got. On the other hand, attacker have no such info, except for length of the sequence.
We use

- Random IV for every message (public).
- AES in CTR mode for encryption (alternatively ChaCha20).
- MAC after encrypt.
- Sequence numbers (inside MAC).
- Key derivation function which generates encryption and MAC keys for both directions.

$$h(k|id)$$

7 Algebra and Number theory

Theorem 7.1 (Invertibility of element). $a \in \mathbb{Z}_n$ *is invertible* $\iff (a, n) = 1$.

Proof. For coprime a, n we can use Extended Euclid algorithm and get the inverse using Bezout coefficients.
Otherwise, no solution exists. □

Definition 7.2.

$$\mathbb{Z}_N^* := \{a \in \mathbb{Z}_N \,|(a,N) = 1\}$$

Group of all invertible elements.

Theorem 7.3 (Lagrange). *If G is finite group, $H \leq G \Rightarrow |H| \mid |G|$.*

Theorem 7.4 (Fermat little). *For p prime, $(x,p) = 1 : x^{p-1} \equiv 1 \mod p$.*

Theorem 7.5 (Euler). *For $N > 1$, $(x,N) = 1 : x^{\varphi(N)} \equiv 1 \mod N$.*

Proof. Consider

$$1 = x^0, x^1, ..., x^k = 1$$

All of them distinct and form a subgroup of $\mathbb{Z}_N^*$.
By Lagrange $k = |H|$ is a divisor of order of the group $\varphi(N)$.

$$\varphi(N) = k \cdot l$$

So:

$$x^{\varphi(N)} \equiv x^{kl} = (x^k)^l \equiv 1^l = 1$$

□

Theorem 7.6 (Chinese Remainder Theorem (CRT)). *For $n_1, ..., n_k$ pairwise coprime, $n := \prod_i n_i$. Then*

$$\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \times ... \times \mathbb{Z}_{n_k} \simeq \mathbb{Z}_n$$

Proof. Proof for $k = 2$, continue by induction.
1) Consider

$$f : \mathbb{Z}_n \to \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}, f(x) = (x \mod n_1, x \mod n_2)$$

Firstly f is injective:

$$f(x) = f(y) \Rightarrow f(x-y) = (0,0)$$

Also $(x-y)$ is divisible by both $n_1, n_2 \Rightarrow (x-y)$ is divisible by $n_1 \cdot n_2 = n \Rightarrow y = x$.
Secondly f is bijective as both sets have the same cardinality.
2) Constructive proof: given (a,b), find $f(x) = (a,b)$.
Firslty, find

$$u, v : f(u) = (0,1) \wedge f(v) = (1,0) \Rightarrow f(au + bv) = af(u) + bf(v) = (a,0) + (0,b) = (a,b)$$

$$f(n_1) = (0, c_1)$$

If $c_1 = 1, v = n_1$. Otherwise

$$f(c_1^{-1} \cdot n_1) = c_1^{-1} \cdot (0, c_1) = (0,1)$$

Similarly obtain $f(n_2)$. □

Primality test (sketch)

1. Generate $a \neq 0 \in_R \mathbb{Z}_n$
2. if $(a,n) \neq 1$ (Euclid witness)
   return NO

3. if $a^{n-1} \mod n \neq 1$ (Fermat witness)
   retutn NO
4. YES.

Definition 7.7. Carmichael number: n is composite and

$$a \in \mathbb{Z}_n^* : a^{n-1} \mod n = 1$$

Smallest 561.

Theorem 7.8 (Prime probability). *If n is composite, but not Carmichael $\Rightarrow Pr_a[YES] \leq 1/2$.*

Proof.

$$H := \{a \in \mathbb{Z}_n^* | a^{n-1} \mod n = 1\}$$

Then $H \leq G$, as H is not trivial subgroup. Therefore

$$|H| \leq 1/2 \mathbb{Z}_n^*$$

□

8 RSA

Setup:

- p, q big random primes.
- $n := pq$
- $\varphi(n) = (q-1)(p-1)$
- $(e, \varphi(n)) = 1$ is an encryption exponent
- select decryption exponent d s.t $ed = 1 \mod (\varphi(n))$
- (e, n) is an enc key, (d, n) is an dec key

Then we encrypt by calculating $E(X) = x^e \mod n$ and decrypt $E(y) = y^d \mod n$. Assuming $(x, n) = 1$, last equality by the Euler theorem

$$(x^e)^d \equiv x^{ed} \equiv x^{1+k\varphi(n)} \equiv x \cdot x^{k\varphi(n)} \equiv x \cdot (x^{\varphi(n)})^k \equiv x$$

Therefore, encryption using RSA is invertible. Relies on the hardness of factorisation. RSA is slow, but polynomial. Which leads to hybrid ciphers. However, we can made some adjustments to improve computations:

- choose small public exponent (3, 17, $65537 = 2^{16} + 1$)
- use CRT for private calculations (compute mod p, mod q and combine using CRT)

Properties 8.1.

RSA

- Commutative: $D_2(D_1(E_2(E_1(x)))) = x$.
- Homomorphic: $E(x_1x_2) = (x_1x_2)^e = x_1^e x_2^e = E(x_1)E(x_2)$.

Example 8.2. RSA usage for **Blind signature**:
Firstly, Alice signs some text. Bob want to sign message secretely from Alice. Steps:

1. Bob generates $b \in_R \mathbb{Z}_n^*$, sends xb^d to Alice
2. Alice sends back $(xb^d)^e = x^e b$
3. Bob calculates $x^e b b^{-1} = x^e$ and gets the signature done by Alice.

Definition 8.3. Semantic security - any properties of the plain text cannot be efficienlty computed from cipher text.

Attacks on RSA

1. If $x < \sqrt[e]{n} \Rightarrow$ decrypt is e-th root in $\mathbb{Z}_n$, not that hard.
2. If anybody knows $\varphi(n) \Rightarrow$ he can factorise n.
3. If anybody knows $e, d \Rightarrow$ he can factorise n.
4. Wiener. If $d < \sqrt[4]{n} \Rightarrow$ d could be obtained from e probabilistically.
5. Meet in the meedle: we know $c = m^e$, e is also public. Try random small u, v to get a collision. Then
$$u^e \equiv cv^{-e} \iff u^e v^e = m^e \iff (uv)^e \equiv m^e \Rightarrow ev = m$$
6. similar massages: $m, (m+\delta)$.
$$c = m^e \wedge c' = (m+\delta)^e \iff P(m) = m^e - c \wedge P'(m) = (m+\delta)^e - c'$$
Therefore, m is the root of 2 polynomials. If e is small $\Rightarrow P, P'$ have small degree and m is a root of $gcd(P, P')$. Which is a linear polynomial with high probability.
Alternatively, the attacker can guess δ.
7. p and q are close to each other $\Rightarrow$ easier factorisation of n.
8. single message encrypted with the same e but different mod. Ex: $e = 3, n_1, n_2, n_3$ are 3 different residuals
$$\begin{aligned} x^3 &= y_1 \mod n_1 \\ x^3 &= y_2 \mod n_2 \\ x^3 &= y_3 \mod n_3 \end{aligned}$$
So,
$$x < \min(n_1, n_2, n_3) \Rightarrow x^3 < n_1 n_2 n_3$$
and x^3 can be calculated using CRT. Then, cubic root.
Solutions: high e or randomize messages.

9. Not 100% Semantic secure, as RSA leaks Jacobi Symbol

$$\left(\frac{x}{p}\right)\left(\frac{x}{q}\right)$$

Which is approximately 1 bit of information.

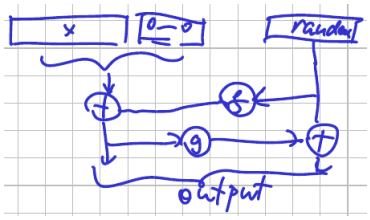
However, computing parity of x from $E(x)$ is equivalent to full decryption.

To avoid many pitfals of RSA design, we can use Padding scheme.

- PKCS v1.5: mix message and random bits:

There exists a oracle padding attack, Bleichenbacher.

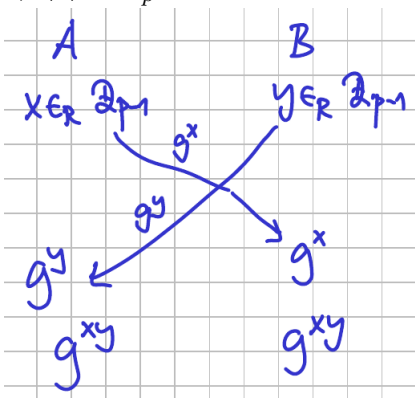
- PKCS v2: add 2 hash functions and XORs (2 Feistel networks)



Proved to be secure against oracle padding attack.

8.1 Diffie-Hellman key exchange protocol

Parameters:
1) p is a prime
2) $\langle g \rangle = \mathbb{Z}_p^*$ generator



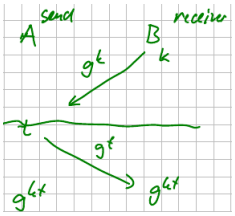
Problems:

- Man-in-the-middle could alter messages. Solution: sign the result
- Attack on parameters: replace g by g^k which generates only a subgroup of $H \leq \mathbb{Z}_p^*$ Solution: check the parameters.
- Powering attack: Mallory replaces g^x sent by Bob by g^{kx}, similarly for Alice. Computed signature on both sides still matches, however we are in the smaller subgroup again. Whick makes finding the generator much easier.

  Solution: sign the whole communication.

- D-H leaks whether g^{xy} is a quadratic residue, < 1 bit leak
- Common trick to fix subgroup attacks: choose safe prime $p = 2q - 1$. From Lagrange theorem, there are only 2 non trivial subgroups, larger of them is subgroup of quadratic residues. Easy to check.

El Gamal cipher based on D-H



- Params: prime p, $\langle g \rangle = \mathbb{Z}_p^*$ generator
- Keys: $k \in_R \{0, ..., p-2\}$ secret. $h = g^k \mod p$ public
- Enc: $t \in_R \{0, ..., p-2\}$

  $$s = h^t = g^{kt} \mod p, y = xs$$

  Send (g^t, y).
- Dec: Calculate $(g^t)^k$

  $$x = y \cdot s^{-1} \mod p$$

Note 8.4. We can also use different Algebraic structures with even harder descrete log, e.g. Elliptic curves.

9 Practical

9.1 Secure channel

Assume: everybody has his secret key + everybody else's public key. How to init secure channel?

1st version

1. Exchange nonces
2. A generates master secret M randomly
3. A encrypts M using B's public key, sends to B
4. Both signs steps 1-3 and exchange signatures (to ensure nobody's playing A and B role in the middle)

2nd version D-H exchange for M. This option is better, since it gives us *forward security*.
If attacker decrypts M in 1st version he can decrypt all historic communication. For D-H master key is never sent over the network (only the signature), so the attacker does not have access to plain text.

9.2 Practical issues of secure SW

- We do not know how to design secure SW. (complexity is the worst enemy)
- We do not know how to implement secure SW
- Debugging by testing is not enough. As it covers only known cases 1) random input 2) prepared structure. Coverage analysis can help. Fuzz testing (random modifications of the input)
- We need proofs (model checking), code reviews etc.
- Using C is an issue (buffer overflows, strlncpy etc.)
- Using languages other than C can be an issue (timing attacks by execution difference, complexity of compilers etc.) Rust could be good option, or combination of Python + C.
- Too many dependencies (libraries often designed for performance and usage but not security, firmware, HW)
- Real attacker is much more powerful than in theory.

Real attacker types:
1) Remote (via network). Can measure timing (side channel). For example if we compare MAC using memcmp in C (compares byte by byte in memory and stops at first difference). An attacker can modify signature and measure timings (or averages if difference is small). He needs $256 \cdot 32 \approx 8k$ and gets all bytes.
Solution: compare using XOR ($sign[i] \oplus s[i]$). Make sure that compiler does not optimize this code!
Another attack is file type interpretation. In Unix, no metadata is stored in disk. On the other side, Windows uses file extensions to choose program that opens file (can be misleaded by e.g. *file.exe.jpg*). We can also create file which is a valid JPG and Java Byte code at the same time.
Another issue are ambiguities:

- Null terminated string vs Pascal strings (length + string). Copying 1st type to second can be fatal.
- In UTF-8, char 'a' has 1 byte code (ASCII), 2,3,4-byte codes. Only 1 byte code is valid according to the standard. Some implementations, however, tolerate all possibilities.
- JSON parsing can lead to differences as no fixed standard exists. Solution: use more strict subset of JSON.
- Differences in XML comments parsing led to problems in iOS $\Rightarrow$ avoid XML in security critical programs.

DDOS attack: flood server by messages.
2) Attacker in the same room. He is able to:

- Measure power consumption (can help to distinguish $x^e \mod m$ vs $x^2 x$).
- Measure sound (depends on load), keyboard clicks etc.

- Thermal (keys are wormer after press)
- Radio emissions (voltage spikes), magnetic emissions.

3) Attacker with access to PC (mo way to defend)
Can install a spy bug (HW device), e.g. keyboard that logs keystrokes. Or even extract memory modules, put the into freezer and extract data.
4) Program on the machine (e.g. JS)
Mainly exploits HW side effects (e.g. shared caches) or HW bugs (meltdown, spectra etc.). Cache side effects could be used to extract AES key from other program.

Note 9.1. Computers designed for security use simple CPUs.

9.3 Storing secrets

1) Keys in memory. Can be stored to disk (swap). Can be sent over network by mistake (e.g. data remained in non initialized variable).
Precaution:

- Erase secrets when not needed
- Linux mlock to disable swapping pages
- Disable core dumps
- Split secrets (store in 2 places)

2) Data on disk (how to secure erase?)
Basic delete just erases the metadata and not the information. Override does not work, as disk optimization can move new information to different place in memory (also defragmentation). Even overriding the whole disk does not help because of bad blocks (they remember some data).
Also, low level details can tell if bit was 1/0 (voltage and other parameters).

Note 9.2. Nobody knows how to erase data on SSD because of optimization algorithms.

Although, there are disks, that support secure erase function, manifacturers are cheating and do not implement it property. Quality of randomness is also an issue.

Dedicated secure HW Chip cards, crypto modules.
Even though, companies produce such hardware, we cannot trust them fully. As there were examples of such companies working for CIA and other agencies.

Keeping state

- Keep nonces
- Keep RNG state (after reboot, crash or from backup)
- First boot (many WiFis are identical, can be cracked at first boot). Solution: distribute different states among identical devices.

9.4 Practical recommendations

- Everything can be compromised.
- Ideas real security.
- Slow down the attacks in order to catch attackers easier.
- Make an attack visible (logs, monitoring etc.).
- Increase cost of attacks by making it > cost of secrets.