Cryptography

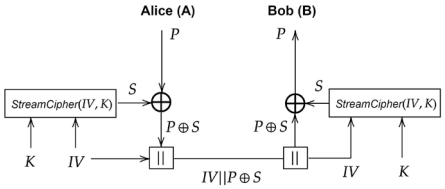
One-Time Pad: Two-Pad Attack

- Reuse of same key stream leaks relation between plaintexts: C1 = P1 ⊕ K, C2 = P2 ⊕ K ⇒ C1 ⊕ C2 = P1 ⊕ P2
- If one plaintext is known or guessable, the other can be recovered.
- Equal plaintexts produce equal ciphertexts (pattern leak).
- Rule: Never reuse key stream (also applies to stream ciphers).

Stream vs Block Ciphers

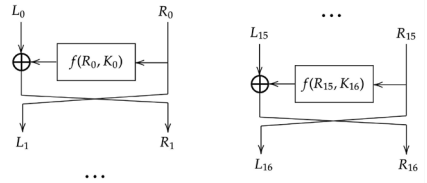
Stream Ciphers

- Generate pseudo-random key stream S = StreamCipher(IV,K).
- Encryption: C = P ⊕ S.
- Fast, bit/byte oriented, low latency.
- Fatal if nonce reused → same two-pad attack.



Block Ciphers

- Operate on fixed block size (AES: 128-bit).
- Modes of operation semantics: ECB – deterministic, pattern leaks, avoid. CBC – uses random IV, needs MAC for integrity. CTR – turns block cipher into stream, needs unique nonce. GCM – AEAD mode (confidentiality + integrity).
- Pros: hardware support, flexible AEAD modes.
- Cons: padding issues (CBC), IV misuse.



Theoretical Properties

Symmetric Encryption

- Provides confidentiality.
- Security defined by indistinguishability (IND-CPA / IND-CCA).
- Key space 2^k; brute force cost ≈ 2^(k-1).

Asymmetric (Public-Key)

- Uses key pair (pk, sk).
- Enables key exchange, encryption without pre-shared secret, and signatures.
- Slower; usually used only for key establishment.

Hash Functions

- Map variable input to fixed n-bit output.
- Properties: Preimage resistance – hard to find input for given hash. Second-preimage resistance – hard to find different input with same hash. Collision resistance – hard to find any two inputs with same hash (≈2^(n/2) effort).
- Used for: integrity, digital signatures, password hashing, HMAC, commitments.

Problem Solved by Asymmetric Cryptography

- Addresses the key distribution problem: allows secure communication without pre-shared secret.
- Anyone can encrypt using public key; only private key holder can decrypt.
- Enables digital signatures, hybrid encryption(Use PK share symmetric key).

Confidentiality, Integrity, Authentication

Confidentiality

- Only authorized parties can read the message.
- Achieved by encryption (e.g., AES, ChaCha20).

Integrity

- Message cannot be modified undetected.
- Achieved by MAC, HMAC, AEAD, or digital signatures.

Authentication

- Entity authentication: verify who is communicating (e.g., challenge-response).
- Message authentication: verify origin and integrity of message (MAC or signature).

Relationships

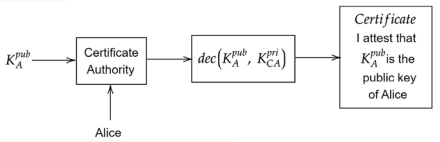
- Encryption alone → confidentiality only.
- Signature or MAC → integrity + authentication.
- AEAD (AES-GCM, ChaCha20-Poly1305) → confidentiality + integrity together.
- Signatures do not hide message content.

Digital Signatures

Security Properties

- Authenticity – verifies identity of signer (via certified public key).
- Integrity – any change in message breaks verification.
- Non-repudiation – signer cannot deny valid signature.

Structure



1. Alice gets identified by the CA
2. Alice's public key K^{pub}_A gets signed by the (trustworthy) CA using the CA's private key K^{pri}_{CA}
3. The signed public key K^{pub}_A is linked to Alice and the CA provides a certificate
4. Anyone that wants to confirm the certificate can check the CA's signature by encrypting the certificate using the CA's public key K^{pub}_{CA}

Cryptographic Implementations:

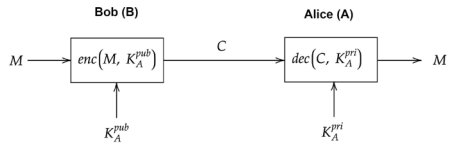
Performance parameters for hardware design

- Area (silicon used), Throughput (data per time), Power (peak), Energy (per task), Time-to-Market (design time), Latency (time per item)

Important parameters when implementing crypto in HW/SW

- From HW perspective: fast speed (sometimes), low latency, low area, low power, low energy.
- From SW perspective / benchmarking: know the platform (word size, available instructions), measure clock cycles, memory occupation, power, energy; aim for fair, reproducible results.

Public-Key Encryption / Decryption



Software implementation: LUTs vs compute-on-the-fly

- **LUT (tables):** implement S-boxes and even whole round functions as look-ups → fewer instructions, higher speed; costs more memory. Best when code footprint and cache allow.
- **Compute on the fly:** cheaper in memory, portable across platforms; may increase cycles (consider shift/rotate costs on target CPU). Balance using platform knowledge (word size, instruction costs).
- **Practical approach:** profile on the target, report cycles and memory; pick the mix (partial tables + computed steps) that minimizes cycles under your memory budget.

Crypto algorithm selection nowadays (what guides the choice)

- Start from specifications: algorithm standard and system requirements drive the design flow.
- Then: Use standardized, publicly scrutinized algorithms (e.g., AES, RSA/ECC), then match to constraints (area, power, latency, throughput).
- Trend facts from slides: RSA (large keys) is slower; many systems moved to ECC; post-quantum migration is underway (lattice-based). Choose based on required security level and platform performance.

Cryptanalysis

What is a fault attack / DFA (high level)

- Attacker injects faults during computation (active attacker), obtains a fault-free ciphertext C and a faulted ciphertext C' for the same plaintext P, and uses C ⊕ C' (and how the fault propagates) to recover key material (divide-and-conquer on S-boxes / round parts).

Countermeasures (fault attacks)

- Redundancy / repetition: run operation twice and compare outputs (check mismatch → alarm).
- Error detection / correction for internal states.
- Sensors for voltage/clock/laser anomalies.
- **Note:** redundancy or comparisons themselves can be faulted (so use hardened checks / multi-layer defenses).

Timing attacks — idea & countermeasures

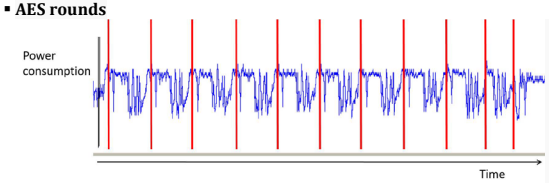
- **Idea (simple):** If execution time depends on secret data (branches, memory/cache behavior), attacker measures time (locally or remotely via RTT) and infers secret bits (e.g., PIN checking with early exit reveals prefix bits).

Practical countermeasures (timing)

- Constant-time implementations: remove data-dependent branches and memory access patterns.
- Avoid LUTs or ensure accesses do not depend on secret (compute S-box algebraically instead of table lookup; or use platform features).
- **Hardware fixes:** special hardware that enforces constant-time memory access (slower).
- **Random delays:** can increase noise but do not guarantee security (CLT reduces effectiveness with many samples).

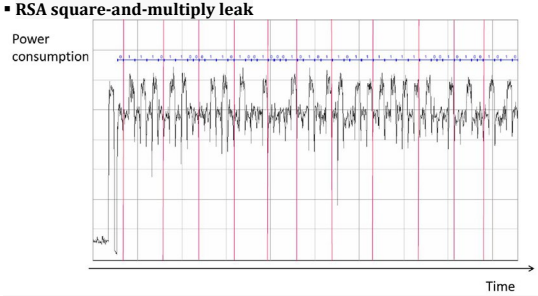
Side channel attacks:

- **AES rounds**



10 repeating patterns thus it's AES-128, last round smaller due to the lack of MixColumns

- **RSA square-and-multiply leak**

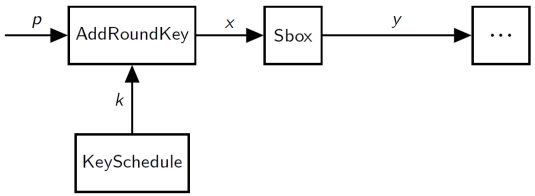


Modular exponentiation does square every loop and multiply only when exponent bit = 1. Presence/absence of the multiply shows up as longer execution and different power patterns(Timing+Power pattern attack.)

Correlation Power Analysis (CPA) Attack

Step 0: To perform CPA we must know the cipher' s structure

- All values (p, k, x, y) have size of 8 bits
- x = p ⊕ k
- y = S(x)



Step 1: Choose an intermediate value v of the cipher to attack
v must be a function of the plaintext and the key i.e. v = f (p, k)
A common choice for v is the Sbox output i.e. v = y = S(p ⊕ k)
Note that recovering y is equivalent to recovering k since: x = S^{-1}(y), the Sbox S is one-to-one and p is known, thus k = x ⊕ p

Step 2: Measure the power consumption of the device

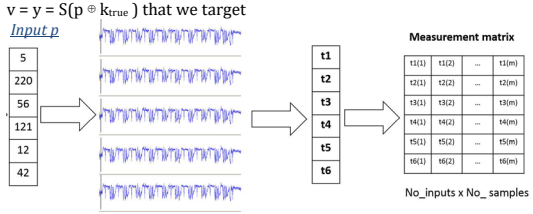
- In the measurement setup we generate randomly n 8-bit plaintexts pi

Typically the number of traces n is large i.e. thousands to millions

- Store the plaintexts in the input vector p = [p1, p2, p3, . . . , pn]
- While encrypting every pi , digitized signal over time i.e. the trace ti
- Every trace ti contains m time samples i.e.

$$\mathbf{t}_i = \left[t_i[1],\ t_i[2],\ t_i[3],\ \ldots\ t_i[m] \right]$$

- Capturing n power traces with m time samples per trace results in the measurement matrix M
- At some point in time the device should process the intermediate v = y = S(p ⊕ k_{true}) that we target



Step 3: Compute the hypothetical intermediate values

- In our device v = y = S(in ⊕ ktrue), but k_{true} is unknown
- For any plaintext pi we can compute the value yi for all possible key candidates k ∈ {0, 1, . . . , 255}, only **1 key (1 col) correct**

Value-prediction matrix

	k=0	k=1	...	k=255
in1	Sbox(in1 XOR 0)	Sbox(in1 XOR 1)	...	Sbox(in1 XOR 255)
in2	Sbox(in2 XOR 0)	Sbox(in2 XOR1)	...	Sbox(in2 XOR 255)
in3	Sbox(in3 XOR 0)	Sbox(in3 XOR 1)	...	Sbox(in3 XOR 255)
in4	Sbox(in4 XOR 0)	Sbox(in4 XOR 1)	...	Sbox(in4 XOR 255)
in5	Sbox(in5 XOR 0)	Sbox(in5 XOR 1)	...	Sbox(in5 XOR 255)
in6	Sbox(in6 XOR 0)	Sbox(in6 XOR 1)	...	Sbox(in6 XOR 255)

No_inputs x No_keys

Step 4: Apply the leakage model

- We map the hypothetical intermediate values to hypothetical power consumption values, producing the power-prediction matrix
- A common choice for leakage model is **Hamming weight (HW)** of a value

Power-prediction matrix *PP*

$$\begin{matrix} \text{no. traces} \\ \\ \\ \\ \end{matrix} \begin{bmatrix} HW(S(p_1 \oplus 0)) & HW(S(p_1 \oplus 1)) & \ldots & HW(S(p_1 \oplus 255)) \\ HW(S(p_2 \oplus 0)) & HW(S(p_2 \oplus 1)) & \ldots & HW(S(p_2 \oplus 255)) \\ HW(S(p_3 \oplus 0)) & HW(S(p_3 \oplus 1)) & \ldots & HW(S(p_3 \oplus 255)) \\ \vdots & \vdots & \ddots & \vdots \\ HW(S(p_n \oplus 0)) & HW(S(p_n \oplus 1)) & \ldots & HW(S(p_n \oplus 255)) \end{bmatrix}$$

no. key candidates

Step 5: Compare power-prediction matrix PP to measurement matrix M

- Use Pearson's correlation coefficient: uncorrelated columns ρ a,b ≈ 0, correlated columns ρ a,b ≈ ±1, CPA use absolute correlation | ρ a,b|
- Compare columns of power-prediction PP with measurement M

$$\rho_{\mathbf{a},\mathbf{b}} = \frac{\sum_{i=1}^{n}\left(a_i - \frac{1}{n}\sum_{i=1}^{n}a_i\right)\left(b_i - \frac{1}{n}\sum_{i=1}^{n}b_i\right)}{\sqrt{\sum_{i=1}^{n}\left(a_i - \frac{1}{n}\sum_{i=1}^{n}a_i\right)^2}\sqrt{\sum_{i=1}^{n}\left(b_i - \frac{1}{n}\sum_{i=1}^{n}b_i\right)^2}}$$

Hamming weight (HW) and Hamming distance (HD) models

- **Hamming weight (HW)** of value v: number of '1' bits in v. Example: HW(10011)=3.
- HW often models bus precharge or software register loads (microcontrollers) where power correlates with number of ones.

- **Hamming distance (HD)** between two values a,b: number of bit flips to go from a to b (HD(a,b)=HW(a⊕b)). Example: HD(10011,01101)=4.
- HD models dynamic switching (register transitions) and is common for hardware (FPGA/ASIC).
- **Use in attacks:** Map hypothesized intermediate values to HW/HD predictions and correlate with measured traces (see CPA next).

Intrusion Detection Systems:

Confusion Matrix and Basic Metrics

- True Positive **TP** = P('alert'|'intrusion') = P(A|I)
- True Negative **TN** = P('no alert'|'no intrusion') = P(¬A|¬I)
- False Positive **FP** = P('alert'|'no intrusion') = P(A|¬I)
- False Negative **FN** = P('no alert'|'intrusion') = P(¬A|I)

IDS Decision \ Reality	Intrusion	No Intrusion
Intrusion Alert	True Positive	False Positive
No Intrusion No Alert	False Negative	True Negative

- Base rate parameter **B =** P(I) i.e. the probability of an intrusion
- Bayesian detection rate (positive predictive value) **PPV**
  - PPV close to 1: after we see an alert, since an intrusion is likely
  - PPV close to 0: can ignore the alert, since an intrusion is unlikely

$$PPV = P(I|A) = \frac{P(I,A)}{P(A)} = \frac{P(A,I)}{P(A)} = \frac{P(A|I)P(I)}{P(A)} =$$

$$\frac{P(A|I)P(I)}{\sum_{\mathbf{X}\in\{I,\neg I\}} P(A,\mathbf{X})} = \frac{P(A|I)P(I)}{\sum_{\mathbf{X}\in\{I,\neg I\}} P(A|\mathbf{X})P(\mathbf{X})} = \frac{P(A|I)P(I)}{P(A|I)P(I)+P(A|\neg I)P(\neg I)} \iff$$

$$PPV = \frac{TP * B}{TP * B + FP * (1-B)}$$

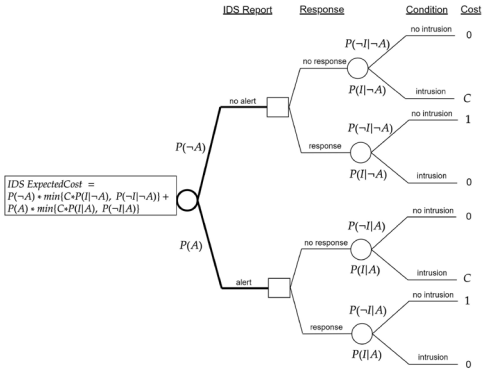
- Similarly, negative predictive value **NPV** = P(¬I |¬A)

Base Rate Fallacy (PPV can be low even with good TP)

- With rare intrusions (small B), even small FP dominates alerts.
- Example: TP=0.99, FP=0.01, B=0.001 → PPV ≈ (0.99·0.001)/(0.99·0.001 + 0.01·0.999) ≈ 0.09 (≈9% of alerts are real).
- Takeaway: Good IDS must good at suppressing false alerts, low FP

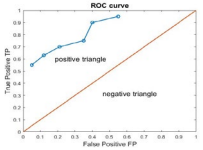
Decision Trees: Roll-Back (Expected-Cost) Analysis

- Leaves to Parent node, best operating point for IDS minimizes cost



Receiver Operator Characteristic (ROC) Curve

- 2d graph showing tradeoff between **TP (y-axis)** and **FP (x-axis)** in [0, 1]
- **low TP / low FP** 'cautious': will raise an alert if strong evidence
- **high TP / high FP** 'relaxed': will raise an alert if weak evidence
- Goal is high TP low FP, above the line better than random guess



Distance Based Outliers (DBO)

- Definition: A datapoint o in D is a DB(p, d) outlier if at least fraction **p** of the datapoints in D lie in distance greater than **d** from o
- **Construction**
  Step 1: Construct the following set S:
  S = {x in D s.t. dist(o, x) <= d} dist(o, x) = Euclidean distance between o,x
  Step 2: Compute the set cardinality |S|
  If **|S| <= (1 - p)|D|** then the datapoint o is an outlier
  If **|S| > (1 - p)|D|** then the datapoint o is normal behavior
- Simplify the anomaly detection problem and use distance metric

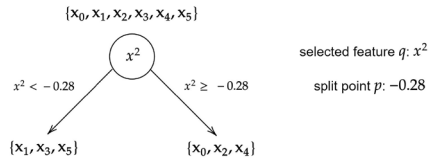
Isolation Tree (Isolation Forest)

- **Construction**

Assume that we have the following training dataset with:
- 6 datapoints stored in vectors {x0, x1, x2, x3, x4, x5}
- 5 features [x0, x1, x2, x3, x4] for each datapoint xi

$$\mathcal{D} = \begin{bmatrix} \mathbf{x_0}^\top \\ \mathbf{x_1}^\top \\ \mathbf{x_2}^\top \\ \mathbf{x_3}^\top \\ \mathbf{x_4}^\top \\ \mathbf{x_5}^\top \end{bmatrix} = \begin{bmatrix} 4.74 & 0.71 & -0.26 & 2.79 & 5.11 \\ 4.71 & 0.75 & -0.30 & 2.50 & 5.58 \\ 4.72 & 0.82 & -0.28 & 2.61 & 5.26 \\ 4.66 & 0.72 & -0.31 & 2.62 & 5.30 \\ 4.80 & 0.71 & -0.27 & 2.71 & 5.33 \\ 4.81 & 0.77 & -0.29 & 2.72 & 5.40 \end{bmatrix}$$

Step 1: Randomly select a feature q from {x0, x1, x2, x3, x4} e.g. q = x2
Step 2: Randomly select a split point p between max and min of q
e.g. max(x2) = -0.26 and min(x2) = -0.31 p = [-0.31,-0.26], pick p = -0.28
Step 3: Partition dataset D to the left and right sets D_l and D_r such that: D_l = {x in D s.t. q < p} and D_r = {x in D s.t. q > p} refer as **isolation tree** (iTree)



Step 4: The construction of the iTree continues recursively
- **iForest idea**: On average, the path length of anomalies is shorter compared to the path length of normal datapoints

RNG, IoT & OS security

Secure OS: Remove all the unnecessary software; Remove all the unnecessary services; Change default accounts ; Use the least privilege principle; Regularly update; Activate logging

Valid for IoT devices?

- Partly valid, but IoT is different: No antivirus; Limited support lifetime; Poor user interface; The programmers are not battle-hardened; The computer is connected to a physical device.
- Emphasize signed firmware, secure boot, automatic updates/rollback, strong device–hub/vendor auth, network isolation (separate VLAN), default-credential removal, and "hub as firewall/IDS" patterns.

Why IoT protection different from PCs/servers:

- Limited interfaces (no keyboard/screen) → weak configuration & visibility
- No/limited antivirus model and subscription economics
- Constrained resources → constant-time crypto and lightweight stacks; avoid heavy agents.
- Attached to physical processes → safety requires sanity checks (hardware interlocks).
- Cloud/vendor reliance → vendor compromise/abandonment risks; signed firmware a must.
- Usability economics → users rarely update; defaults persist; recovery/reset needs physical controls.

Pseudo Random Number Generator (PRNG)

- **Definition**: state ← F(state, seed) Generated from a seed; Sequences that have the statistical property of a random sequence
- **Useful in security?** A normal PRNG is deterministic and predictable once its seed or internal state is known → not safe for keys or nonces.

True random number generator

[Entropy Source] → [Digitizer / Sampler] → [Post-Processing / Extractor] → [Ring Oscillators] → [Quantum]

User authentication

Password entropy H(X)

$$H(X) = -\sum_{x\in\mathcal{X}} p(x)\log_2 p(x)$$

- Entropy is expressed in bits
- A higher number of bits implies more uncertainty
- When computing the entropy H(X) discard zero probability
- It holds that H(X) ≥ 0 always

Dictionary attacks and Salt values

- **Verification:** store (user, h), where h = hash(password). At login, compute h' = hash(input) and compare h' == h.
- **Method:** Let dictionary D that contains b most common passwords D = {x1, x2, . . . , xb} Attacker can hash them in advance, i.e. create a lookup table
  H = {hash(x1), hash(x2), . . . , hash(xb)}
  Attack amounts to a single table lookup in H thus it is much faster than computing hashes on-the-fly
- **Salt value:** a non-secret value for each password
  Compute: h = hash(x||salt), Store in database: (userid, h, salt);
  To verify a user password x' Compute: h' = hash(x'||salt), Compare: h == h'

Two-factor authentication (2FA): To login now I need something that I know + something that I have.
- e.g. a password together with a passcode generator / a password together with a hardware security token / a password together with mobile authenticator app / a PIN together with a smartcard

Privacy & Anonymity

K-anonymity

- Let a database D with non-sensitive attributes {A1,A2, . . . , An}
- Let a quasi-identifier Q = {Ai , . . . ,Aj} ⊂ {A1,A2, . . . , An}
- Let the database projection D[Q]
  i.e. discard all attributes in D that are not in the Q set

k-**Anonymity**. We say that the database D satisfies k-anonymity with respect to quasi-identifier Q iff every sequence of values in the projection D[Q] appears with at least k occurrences

Benefits
- Ensuring k-anonymity prevents database linkage attacks with a bound of k
- Linking the datasets to external sources will not yield less than k individuals

l-Diversity

- Let a q*-block in anonymized database D*
- The block is **l-diverse** if it contains at least l well-represented values of the
  sensitive attribute S

Example→
- Block 1, with q*=[130**, >=50] has l-diversity of 1
- Block 2, with q* =[140**, <50] has l-diversity of 2
- Block 3, with q* =[130**, <50] has l-diversity of 1

ZIP	Age	Nationality	Condition (S)
130**	≥50	*	Heart
130**	≥50	*	Heart
130**	≥50	*	Heart
130**	≥50	*	Heart
140**	< 50	*	Cancer
140**	< 50	*	Cancer
140**	< 50	*	Heart
130**	< 50	*	Heart
130**	< 50	*	Heart

Benefits
- Large values of diversity will ensure that the homogeneity and background knowledge attacks do not work
- Following the l-diversity principle you try to ensure: β (q,s,D*) ≈ α (q,s)

Entropy l-diversity

A database is entropy l-diverse if for every q*-block it holds that:

$$\sum_{s\in S} p_{(q^*,s)} \log_2(p_{(q^*,s)}) \geq \log_2(l)$$

where $p_{(q^*,s)} = \frac{n_{(q^*,s)}}{\sum_{s'\in S} n_{(q^*,s')}}$ 单个 / 总数

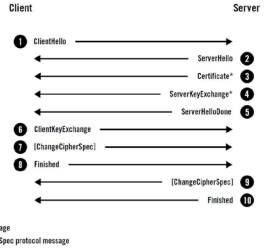
Security Protocols

MITM attack pattern (relay + substitution)

- Intercept A↔B handshake.
- Replace public keys/certificates or swap nonces.
- Replay previously observed signed values when freshness is not checked.
- If either party doesn't validate a certificate chain or check that the key belongs to the claimed identity, attacker succeeds.

TLS Protocol: Handshake protocol (public-key operations)+ Record protocol (symmetric)

Handshake protocol



- A full handshake starts with ClientHello where the Client submits its connection and security parameters to the Server
- The Server replies with ServerHello that selected the parameters
- The Server sends a signed Certificate for authentication and the Client verifies the signature
- The Client and Server use public key cryptography to exchange keys. The Server sends the needed information via ServerKeyExchange and notifies that is done.
- The Client sends the needed information via ClientKeyExchange, derives the premaster secret and the master secret
- Now the Client switches encryption on and notifies the Server via ChangeCipherSpec
  To verify the integrity of the hanshake so far, the Client computes a MAC(handshake messages) and sends it to the Server via Finished
- The Server derives the premaster secret and the master secret, switches encryption on and notifies the Client via ChangeCipherSpec
- To verify the integrity of the hanshake, the Server computes a MAC(handshake messages) and sends it to the Client via Finished
- A TLS session between Client-Server may now begin using a symmetric algorithm for bulk encryption of application data

Record Protocol
- Record protocol transports and encrypts all connection messages
- The record keeps also a unique 64-bit sequence number to avoid replays