Table of contents

Key Establishment Protocols (KEP)

KEP Definition and Properties

Definitions and Classification

Key Establishment Protocol (KEP): A mechanism enabling entities to share a secret for their cryptographic exchanges.

Two Types:

- **Key Transport Protocol (KTP):** One entity creates and transmits the key.
- **Key Agreement Protocol (KAP):** Entities jointly derive the key.

Temporal Classification:

- **Pre-distribution:** Keys determined in advance.
- **Dynamic Key Establishment (DKE):** Keys change with each execution.

Key Establishment Protocols

Key Agreement

Key Transport

Symmetric + pre-dist.

Symmetric + DKE

Symmetric + DKE

Asymmetric + pre-dist.

Asymmetric + DKE

Asymmetric + DKE

Original Text

A **key establishment protocol (KEP)** is one that provides the involved entities with a **shared secret** (a key) to serve as the basis for subsequent cryptographic exchanges. The two variants of KEPs are **key transport protocols (KTP)** and **key agreement protocols (KAP)**.

- A **key transport protocol (KTP)** is a mechanism allowing one entity to **create a secret key and transfer it** to its correspondent(s).
- A **key agreement protocol (KAP)** is a mechanism allowing two (or more) entities to **derive a key** from information specific to each entity.

Key pre-distribution schemes are those where the keys used are entirely **determined a priori** (e.g., from initial calculations).
Dynamic key establishment schemes (DKE) are those where the keys **change for each protocol execution**.

Quick Review

KEP: Protocols to establish a shared secret.

- **KTP:** Key transport
- **KAP:** Mutual key agreement
- **Pre-distribution** vs. **DKE** (dynamic)

KEP Properties

Authentication Properties:

- **Implicit key authentication:** Assurance that only the correspondent can access the key (without proof of possession).
- **Key confirmation:** Assurance that the correspondent effectively possesses the key.
- **Explicit key authentication:** Implicit + confirmation.
- **Authenticated KEP:** A KEP providing key authentication.

Temporal Security Properties:

- **Perfect Forward Secrecy (PFS):** Compromise of long-term keys does not reveal past session keys.
- **Future Secrecy:** Future keys are protected even if long-term keys are compromised (by a passive attacker).
- **Deniability/Repudiability:** Inability to prove participation to a third party (like Zero-Knowledge).

Types of Attacks:

- **Passive attack:** Recording and analyzing exchanges.
- **Active attack:** Modifying or injecting messages.
- **Known-key attack:** Exploiting a compromised session key to attack future keys.

Original Text

Key Establishment Protocol Properties:

- **Implicit key authentication** (or key authentication): A property by which an entity is assured that only its correspondent(s) can access a secret key. However, this does not specify anything about actually possessing the key.
- **Key confirmation:** A property allowing an entity to be sure that its correspondents are in possession of the generated session keys.
- **Explicit key authentication:** = implicit key authentication + key confirmation.
- An **authenticated KEP** is a KEP capable of providing key authentication.

Attacks:

- A **passive attack** attempts to break a cryptographic system by **recording and analyzing** exchanges.
- An **active attack** involves an adversary who **modifies or injects** messages.

- A protocol is said to be vulnerable to a **known-key attack** if, when a previous session key is compromised, it becomes possible to: (a) compromise future keys via a passive attack and/or (b) mount active attacks aiming at identity impersonation.

Modern Properties:

- **Perfect Forward Secrecy (PFS)** is a characteristic that guarantees the **confidentiality of past session keys** even if long-term keys (e.g., the recipient's private key) are compromised.
- **Future Secrecy:** The protocol guarantees the **security of future exchanges** (future session keys are protected) even if long-term keys are compromised by a passive attacker.
- **Deniability / Repudiability:** Similar to Zero-Knowledge authentication protocols, this allows entities to ensure exchange authentication without providing information that would allow proving their participation in the cryptographic exchange to a third party.

Quick Review

Authentication:

- **Implicit:** Only the correspondent can access the key
- **Key confirmation:** Proof of possession
- **Explicit:** Implicit + confirmation

Security:

- **PFS:** Past keys protected if compromise occurs
- **Future Secrecy:** Future keys protected
- **Deniability:** Participation not provable

KAP

Symmetric

with Pre-distribution

Trivial Case:

For n users with a Key Distribution Center (KDC):

1. KDC generates $\frac{n(n-1)}{2}$ different keys (one per user pair).
2. KDC distributes $n-1$ keys to each user via a confidential channel.

Advantages:

- Information-theoretically secure against user collusion.

Disadvantages:

- $O(n^2)$ storage complexity for the KDC.
- $O(n)$ keys per user.
- Not scalable.

Original Text

Symmetric KAP with Pre-distribution – Trivial Case
Given n users with a **Key Distribution Center (KDC)**.
A trivial symmetric KAP with pre-distribution can be constructed as follows:

(1) The KDC generates $n(n-1)/2$ different keys (one different key for each user pair).

(2) The KDC then distributes the keys via a **confidential and authentic channel**, giving $n-1$ keys to each user.

If the KDC generates the keys in a truly **random** manner, this system is **information-theoretically secure against user collusion** (even if $n-2$ users collude, they cannot find the key of the other two) by protocol construction.
Problem with this protocol:

- $O(n^2)$ **key storage** for the KDC.
- $O(n)$ **secret keys exchanged** for each entity.

Quick Review

Trivial symmetric KAP:

- $n(n-1)/2$ keys for n users
- Information-theoretically secure
- Problem: $O(n^2)$ storage

---

with Dynamic Key Establishment (DKE)

Simple Example

Initialization: A and B share a long-term key S.

Protocol:

1. $A \rightarrow B : r_a$ (random number)
2. $A \leftarrow B : r_b$ (random number)
3. Session key: $K := E_S(r_a \oplus r_b)$

Properties:

- Entity authentication
- Implicit key authentication
- Key confirmation
- Perfect Forward Secrecy

AKEP2 (Authenticated Key Exchange Protocol 2)

Initialization: A and B share S (for MAC) and S' (for session key).

Protocol:

1. $A \rightarrow B : r_a$
2. $A \leftarrow B : T = (B, A, r_a, r_b), h_S(T)$
3. $A \rightarrow B : (A, r_b), h_S(A, r_b)$
4. Session key: $K := h'_{S'}(r_b)$

Properties:

- Mutual entity authentication
- Implicit key authentication
- Key confirmation
- Perfect Forward Secrecy

Note: The key depends only on B and the long-term key S'!

> Original Text
>
> **Symmetric KAP with Dynamic Key Establishment**
> These methods allow the involved entities to derive **short-term keys** (typically session keys) from **long-term secrets**, which, for these protocols, are symmetric keys.
> **Intuitive Example:**

(Initialization): A and B share a long-term symmetric key S.

(1) $A \to B : r_a$; A generates a random number and sends it to B.

(2) $A \leftarrow B : r_b$; B does the same.

A and B compute the session key: $K := E_S(r_a \oplus r_b)$.
Properties: - **Entity authentication:** NO: By protocol construction, the r_i can be sent by any entity. - **Implicit key authentication:** YES: Only entities sharing the long-term symmetric key S can access the session key K. - **Key confirmation:** NO: Since the r_i are random, they can be modified by an adversary, preventing A and B from agreeing on the session key K. This would not be detected by the protocol. - **Perfect Forward Secrecy:** NO: If the long-term key S is compromised, all previous session keys can be easily computed by an adversary who recorded all exchanges.
Authenticated Key Exchange Protocol 2 (AKEP2)
(Init.): A and B share two long-term symmetric keys S and S'. S is used to generate MACs $h_S()$ (to ensure integrity and entity authentication), and S' is used for session key K generation.

(1) $A \to B : r_a$; A generates a random number and sends it to B.

(2) $A \leftarrow B : T = (B, A, r_a, r_b), h_S(T)$; B does the same + identities + MAC of everything.

(3) $A \to B : (A, r_b), h_S(A, r_b)$; A verifies the identities and the r_a provided by B; then sends identity + r_b + MAC of everything.

The key is bilaterally computed using a dedicated MAC $h'_{S'}()$: $K := h'_{S'}(r_b)$.
Properties: - **Entity authentication:** YES (mutual, provided by MACs). - **Implicit key authentication:** YES. - **Key confirmation:** NO (no evidence that the key S' is known to the correspondent). - **Perfect Forward Secrecy:** NO (if the key S' is compromised, previous session keys K are also compromised).
The key depends only on B (and the long-term key S'), but the protocol can be easily modified so that the key also depends on A, making it a "true" KAP.

Quick Review

Symmetric KAP with DKE:

- **Simple:** $K := E_S(r_a \oplus r_b)$ – no PFS
- **AKEP2:** Uses MACs for authentication + key derived as $K := h'_{S'}(r_b)$
- No PFS if S' is compromised

Asymmetric with Pre-distribution

Diffie-Hellman

Initialization: Public prime p and generator $\alpha \in \mathbb{Z}_p^*$.

Protocol:

1. $A \to B : \alpha^x \mod p$ (A chooses secret x)
2. $A \leftarrow B : \alpha^y \mod p$ (B chooses secret y)
3. Shared key: $K := \alpha^{xy} \mod p$

- A computes $K := (\alpha^y)^x \mod p$
- B computes $K := (\alpha^x)^y \mod p$

Security:

- Based on the Diffie-Hellman Problem (DHP): Impossible to compute α^{xy} from α^x and α^y.
- Proven result: DHP $\equiv$ DLP.

Man-in-the-Middle (MIM) Attack:

Adversary C intercepts and replaces:

- α^x with $\alpha^{x'}$ to B
- α^y with $\alpha^{y'}$ to A
- C establishes two keys: $\alpha^{xy'}$ with A and $\alpha^{x'y}$ with B

Properties (Unauthenticated DH):

- Entity authentication
- Implicit key authentication (vulnerable to MIM)
- Key confirmation

Symmetric Key Generation:

DH keys are not bit secure.

Solution: Apply a MDC (SHA, MD5) to the entire key K:

$$K_{sym} := \text{SHA-256}(K)$$

Result: KAP with Dynamic Key Establishment.

Original Text

Asymmetric KAP with Pre-Distribution – Diffie-Hellman
Published in 1976, this is the **precursor of asymmetric protocols**.
It allows two entities that have never met to construct a **shared key** by exchanging messages over a **non-confidential channel**.
Protocol:
Initialization: A prime number p is generated, and a generator α of $\mathbb{Z}_p^*$, such that $\alpha \in \mathbb{Z}_{p-1}$. Both numbers are made public.

(1) $A \to B : \alpha^x \mod p$; A chooses a secret $x \in \mathbb{Z}_{p-1}$ and sends the public part.

(2) $A \leftarrow B : \alpha^y \mod p$; B chooses a secret $y \in \mathbb{Z}_{p-1}$ and sends the public part.

A computes the secret key: $K := (\alpha^y)^x \mod p$, and B does the same: $K := (\alpha^x)^y \mod p$.
The **security** of this scheme lies in the impossibility of finding $\alpha^{xy} \mod p$ from $\alpha^x \mod p$ and $\alpha^y \mod p$ (**Diffie-Hellman Problem: DHP**).
Proven result: DHP $\equiv$ DLP.
Diffie-Hellman is **secure** (as much as DHP) against **passive attacks**. In other words, an adversary limited to observing messages cannot find the key K.
This is no longer true for **active attacks**; let's see what C can do by modifying messages: C exchanges secret keys with A and B, respectively: $\alpha^{xy'} \mod p$ and $\alpha^{x'y} \mod p$ (C controls x' and y'). If C re-encrypts each packet it receives with the corresponding public key, the attack will be transparent to A and B.
This attack is called **Man-in-the-Middle (MIM)** and applies to all asymmetric protocols. It is due to the **lack of authentication of public keys**, i.e., when A "talks" to B, it must use the **authentic** public key of B.
Characteristics of Diffie-Hellman (unauthenticated):

- **Entity Authentication:** NO.
- **Implicit key authentication:** NO (due to the MIM attack).
- **Key confirmation:** NO (due to the MIM risk, A cannot be sure that B possesses the shared secret key).

Generating Symmetric Keys from a Diffie-Hellman Shared Key:
The quantities manipulated in DH (notably K) are 512–1024 bits in size (depending on the prime p used).
An intuitive approach to generate smaller symmetric keys (64–128 bits) would be to take a subset of bits from the key K.
Unfortunately, it can be proven that DH keys are not **bit secure**, meaning that subsets of bits (especially the Least Significant Bits) can be computed with an effort not proportional to that required to compute the entire key.

To generate keys securely, it is recommended to **apply a MDC** (like SHA or MD5) to the entire key (possibly chaining MDC applications to obtain successive symmetric keys). This method yields a KAP with Dynamic Key Establishment.

Quick Review

Diffie-Hellman:

- $K := \alpha^{xy} \mod p$ computed independently by A and B
- Secure against passive attacks (DHP $\equiv$ DLP)
- Vulnerable to MIM without authentication
- Generate symmetric keys: $K_{sym} := \text{SHA}(K)$

Asymmetric with DKE

Station to Station Protocol (STS)

Authenticated Diffie-Hellman with digital signatures.

Initialization: Public prime p and generator α. A and B have authentic copies of each other's public keys.

Protocol:

1. $A \to B : \alpha^x \mod p$
2. $A \leftarrow B : \alpha^y \mod p, E_k(S_B(\alpha^x, \alpha^y))$
    - B computes $k := (\alpha^x)^y \mod p$
    - B signs and encrypts the public parts
3. $A \to B : E_k(S_A(\alpha^y, \alpha^x))$
    - A decrypts, verifies B's signature
    - A signs and encrypts in reverse order

Properties:

- Mutual entity authentication (via signatures)
- Implicit key authentication (DHP + signatures prevent MIM)
- Key confirmation (encryption proves possession of k)
- Explicit key authentication (authentication + key confirmation)

- **Perfect Forward Secrecy** (compromise of signature private key does not reveal past session keys)

Efficient Variant: Replace $E_k(S_B(...))$ with $(sig, h_k(sig))$ using MAC instead of symmetric encryption.

Original Text

Asymmetric KAP with DKE – Station to Station Protocol
(Notation) S_A: Signature with A's private key.
(Initialization): (a) Choose a prime p and a generator α of $\mathbb{Z}_p^*$, such that $\alpha \in \mathbb{Z}_{p-1}$. Both numbers are made public (and optionally associated with the participants' public keys).

(b) Participants have access to authentic copies of the correspondents' public keys. Certificates may be exchanged if needed in (2) and (3).

(1) $A \rightarrow B : \alpha^x \mod p$; A generates a secret x and sends the public part.

(2) $A \leftarrow B : \alpha^y \mod p, E_k(S_B(\alpha^x, \alpha^y))$; B generates a secret y and computes the key: $k := (\alpha^x)^y \mod p$ + signs and encrypts the public parts.

(3) $A \rightarrow B : E_k(S_A(\alpha^y, \alpha^x))$; A decrypts by computing $k := (\alpha^y)^x \mod p$, verifies B's signature and the public parts; if OK, A signs + encrypts in reverse order.

B decrypts and verifies A's signature on the public parts. If OK => END.
Characteristics:

- **Entity Authentication:** YES (mutual, provided by signatures).
- **Implicit key authentication:** YES, keys are protected by DHP. The MIM attack is made impossible by signatures.
- **Key confirmation:** YES, both entities prove possession of the key by encrypting quantities with it.
- **Explicit key authentication:** YES: implicit key authentication + key confirmation.
- **Perfect Forward Secrecy:** YES. The only long-term key is the one used for signing/verification. If this key is compromised, past session keys are protected because they are not explicitly exchanged but rather computed via DH.

Obviously, once the signature key is compromised (private key theft), the stated properties no longer hold for future exchanges.
The protocol additionally provides **anonymity** since the parties' identities are protected by k.
Variant: In (2), compute $sig := S_B(\alpha^x, \alpha^y)$, and send $(sig, h_k(sig))$ instead of $E_k(S_B(\alpha^x, \alpha^y))$. Same for (3), observing the protocol's asymmetries.
More efficient solution as it uses a **MAC** rather than symmetric encryption.
Robust and efficient algorithm chosen as the base for key generation in **IPv6**.

Quick Review

Station to Station (STS):

- DH + digital signatures
- PFS: Past session keys protected
- Explicit key authentication
- Used in IPv6

Off-The-Record (OTR) and Signal Protocols

Off-The-Record (OTR)

Protocol (2004) for instant messaging with **repudiability**.

SIGMA Technique (SIGn-and-MAC):

- DH signatures + ephemeral authentication via MAC
- Key Derivation Function (KDF) generates two keys: K_e (AES-CTR encryption) and K_m (MAC)
- Key change per conversation
- **Revelation of previous MAC keys** to ensure repudiability

Simplified Protocol:

1. $A \to B : \alpha^x \mod p$
2. $A \leftarrow B : \alpha^y \mod p, S_B(\alpha^x, \alpha^y), \text{MAC}_{K_m}(B)$
   - B computes $k := (\alpha^x)^y \mod p$
   - $(K_m, K_e) := \text{KDF}(k)$
3. $A \to B : S_A(\alpha^y, \alpha^x), \text{MAC}_{K_m}(A)$

Messages encrypted with K_e.

Signal Protocol

Evolution of OTR for social networks (WhatsApp, Facebook Messenger).

Characteristics:

- Ephemeral asymmetric and symmetric keys
- DH on elliptic curves
- PFS
- Future Secrecy
- Repudiability

Original Text

Off-The-Record (OTR) Protocol
Protocol designed in 2004 to provide authentication and confidentiality services in instant messaging exchanges while preserving the "**repudiable**" nature of an "off-the-record" conversation.
The protocol also satisfies **PFS** and **Future Secrecy** properties in case of long-term key compromise.
It follows the same principles as the Station-to-Station protocol, adding **ephemeral authentication via a MAC** to the DH parameter signatures. This dual technique is called **SIGMA** (SIGn-and-MAC).
It uses a **Key Derivation Function (KDF)** to generate an encryption key (K_e) preserving message confidentiality with AES CTR-mode and a MAC key (K_m) ensuring message origin authenticity.
Each conversation involves a **key change** (new DH parameter exchange) with an additional **plaintext exchange of the MAC keys** (K_m) used in the previous exchange to ensure repudiability!
Schematic OTR Protocol Exchanges:

(1) $A \to B : \alpha^x \mod p$; A generates a secret x and sends the public part.

(2) $A \leftarrow B : \alpha^y \mod p, S_B(\alpha^x, \alpha^y), \text{MAC}_{K_m}(B)$

B generates a secret y, computes the session key $k := (\alpha^x)^y \mod p$, and signs the DH public parts. It then generates keys K_e and K_m via the KDF: $(K_m, K_e) := \text{KDF}(k)$.

(3) $A \to B : S_A(\alpha^y, \alpha^x), \text{MAC}_{K_m}(A)$; A does the same.

Messages are then encrypted with key K_e.
Numerous evolutions of the original OTR protocol have addressed vulnerabilities and improved efficiency.
The Signal Protocol

The Signal protocol is an evolution of the OTR protocol targeting message exchange protection in **social networks**. It also uses ephemeral asymmetric and symmetric keys to ensure **PFS**, **Future Secrecy**, and **repudiability** with DH computations on elliptic curves.
Signal is used to protect messaging platforms such as **WhatsApp** and **Facebook Messenger**, among others.

Quick Review

OTR/Signal:

- SIGMA: signature + MAC
- KDF: generates K_e (encryption) and K_m (MAC)
- Reveals old MAC keys → repudiability
- PFS, Future Secrecy
- Used in: WhatsApp, Facebook Messenger

Secure Remote Password (SRP)

Asymmetric KAP protocol **based on password**, resistant to dictionary attacks.

Initialization:

- $m := 2p + 1$ (safe prime), α generator of $\mathbb{Z}_p^*$
- P: A's password, $x := H(P)$ with H a CRHF
- B stores the **verifier**: $v := \alpha^x \mod m$ (not the password!)

Protocol:

1. $A \to B : \gamma := \alpha^r \mod m$ (A generates secret r)
2. $A \leftarrow B : \delta := (v + \alpha^t) \mod m, u$ (B generates random t, u)
3. A computes $k := (\delta - v)^{r+ux} \mod m$
4. B computes $k := (\gamma v^u)^t \mod m$
5. Key confirmation

Properties:

- Protects passwords from dictionary attacks
- **Verifier-based:** B does not store passwords
- All KEP properties
- Included in SSL/TLS, EAP

Original Text

Asymmetric KAP with DKE – Secure Remote Password Protocol

(a) Let m be a safe prime with $m := 2p + 1$ and p prime.
(b) Let α be a generator of $\mathbb{Z}_p^*$, such that $\alpha \in \mathbb{Z}_{p-1}$.
(c) Let P be A's password and $x := H(P)$ with H a CRHF.
(d) B stores in its password database the **verifier** $v := \alpha^x \mod m$.

(1) $A \to B : \gamma := \alpha^r \mod m$; A generates a secret random number r.

(2) $A \leftarrow B : \delta := (v + \alpha^t) \mod m, u$; B generates a secret random number t and another random number u.

A computes the symmetric key: $k := (\delta - v)^{r+ux} \mod m$.
B computes the symmetric key: $k := (\gamma v^u)^t \mod m$.
A and B prove knowledge of k (key confirmation) in a subsequent exchange.

- SRP **protects passwords** from dictionary attacks.
- B does not store passwords but **verification values** (verifier-based).
- SRP also satisfies **all KEP properties** and is included in many standards (SSL/TLS, EAP, etc.).

Quick Review

SRP:

- Password-based KAP
- B stores verifier $v := \alpha^x$ (not password)
- Resistant to dictionary attacks
- All KEP properties

Attacks on DH and PFS

Logjam Attack (2015):

Active attack enabling:

1. **Downgrade:** Man-in-the-Middle forces use of 512-bit DH group.
2. **Discrete logarithm computation** with Number Field Sieve:

- One-week precomputation for a fixed prime p.
- ~1-minute individual computation after precomputation.

3. **Precomputation reuse:** Many servers use the same p.

Consequence:

State-level actors can compromise PFS on widespread 1024-bit groups.

Solutions:

- Use groups $\geq$ 2048 bits.
- Diversify the primes p used.

Original Text

Recent Attacks on Diffie-Hellman and PFS
In 2015, a group of researchers published a series of attacks on the TLS/SSL protocol allowing:

- Performing a **downgrade** via an active attack called **Logjam**, whereby a man-in-the-middle successfully reduces the Diffie-Hellman group size to **512 bits** for the shared secret key establishment.
- Subsequently computing the **discrete logarithms** of $\alpha^x \mod p$ and $\alpha^y \mod p$ using the **Number Field Sieve** technique.
- For a group based on a fixed prime p, they perform a **precomputation phase** lasting approximately **one week**.
- Once this initial phase is complete, individual logarithm computations take only **about one minute**!
- A statistical observation shows that a significant percentage of servers rely on the **same group** (same prime p), allowing the same precomputation phase to be used to compromise multiple servers.
- One of the conclusions of this research is that **major actors with state-level resources** would currently be able to break PFS when it is based on (very common today…) **1024-bit groups**.

Quick Review

Logjam (2015):

- Downgrade → DH 512 bits

- Precomputation (1 week) + individual computation (1 min)
- Reuse if same p
- States can break PFS on 1024-bit groups

KTP

Symmetric

Trivial Case

Initialization: A and B share long-term key S.

Protocol:

1. $A \to B : E_S(r_a)$
2. Session key: $K := r_a$

Properties:

- Entity authentication
- Implicit key authentication
- Key confirmation (improvement: $E_S(B, r_a)$)
- Perfect Forward Secrecy

Timestamp Variant: $A \to B : E_S(B, t_a, r_a)$ (requires synchronized clocks).

Shamir's No-Key Protocol

DH equivalent in key transport.

Initialization: Public prime p, A and B generate secrets $a, b \in \mathbb{Z}_{p-1}$ with $\gcd(a, p-1) = 1$ and $\gcd(b, p-1) = 1$.

Protocol:

1. $A \to B : K^a \mod p$ (A chooses key K and hides it with ^{a})
2. $A \leftarrow B : (K^a)^b \mod p$ (B exponentiates with b)
3. $A \to B : (K^{ab})^{a^{-1}} \mod p = K^b \mod p$ (A undoes ^{a})
4. B computes K by exponentiating with $b^{-1} \mod (p-1)$

Problem: Vulnerable to Man-in-the-Middle (like DH).

Original Text

Symmetric Key Transport Protocol – Trivial Case
(Init.) A and B share a long-term symmetric key S.

(1) $A \to B : E_S(r_a)$; A generates a random number and encrypts it with S.

The session key used by both entities is $K := r_a$.
Properties:

- **Entity Authentication:** NO.
- **Implicit Key Authentication:** YES (only A and B have access to the key).
- **Key Confirmation:** NO. B cannot be sure that A possesses the key because r_a is a random number. By adding redundancy (e.g., B's identity), B can achieve unilateral key confirmation (and thus, explicit key authentication):

(1)' : $A \to B : E_S(B, r_a)$

- **Perfect Forward Secrecy:** NO.

If B cannot judge the freshness of (1) from r_a alone, it can ask A to include a timestamp, provided synchronized clocks are available:
(1)'' : $A \to B : E_S(B, t_a, r_a)$
Symmetric KTP: Shamir's No-Key Protocol
Number Theory Reminder: If p is prime and $r \equiv t \mod p-1$, then $a^r \equiv a^t \mod p$ $\forall a \in \mathbb{Z}$, and thus: $r \cdot r^{-1} \equiv 1 \mod p-1$ implies $a^{r \cdot r^{-1}} \equiv a \mod p$.
(Init.) (a) Choose and publish a prime p for which it is difficult (by DLP) to compute discrete logarithms in $\mathbb{Z}_p$.

(b) A (resp. B) generates a secret number a (resp. b), such that $\{a, b\} \in \mathbb{Z}_{p-1}$ and $\gcd(a, p-1) = 1$ and $\gcd(b, p-1) = 1$ (so that inverses exist).

(c) For the following, A precomputes $a^{-1} \mod p-1$ and B precomputes $b^{-1} \mod p-1$.

(1) $A \to B : K^a \mod p$; A chooses a key $K \in \mathbb{Z}_p$ and hides it with ^{a}.

(2) $A \leftarrow B : (K^a)^b \mod p$; B exponentiates in turn with b.

(3) $A \to B : (K^{ab})^{a^{-1}} \mod p$; A undoes the exponentiation with $a^{-1} \mod p-1$; but the key remains protected by ^{b}.

B only needs to compute K by exponentiating with $b^{-1} \mod p-1$.
This protocol is the key transport equivalent of Diffie-Hellman (in DH, the key is not transported but bilaterally computed). It suffers from the same problems (notably Man-in-the-Middle) as the latter.

Quick Review

Symmetric KTP:

- Trivial: $K := r_a$ with $E_S(r_a)$
- Shamir: Transport via successive exponentiations
- No PFS

Asymmetric

Needham-Schroeder

Initialization: A and B have authentic copies of each other's public keys.

Protocol:

1. $A \rightarrow B : E_{pub_B}(k_1, A)$
2. $A \leftarrow B : E_{pub_A}(k_1, k_2)$
3. $A \rightarrow B : E_{pub_B}(k_2)$
4. Session key: $K := H(k_1, k_2)$

Properties:

- Entity authentication + implicit key authentication + key confirmation
- Perfect Forward Secrecy (keys entirely determined by exchanged quantities)

Original Text

Asymmetric Key Transport Protocol – Needham-Schroeder Public Key Protocol
(Notation): $E_{pub_E}(X)$ means encrypting with entity E's public key.
(Init): A and B possess an authentic copy (possibly a certificate) of the other's public key.

(1) $A \rightarrow B : E_{pub_B}(k_1, A)$; A generates a random number k_1 + A + Encrypt.

(2) $A \leftarrow B : E_{pub_A}(k_1, k_2)$; B does the same for k_2 + concatenates with k_1 + Encrypt.

(3) $A \rightarrow B : E_{pub_B}(k_2)$; A verifies if k_1 matches, if yes, encrypts k_2; B verifies if k_2 matches with (2).

The key is generated using a cryptographic hash function: $K := H(k_1, k_2)$.
Characteristics:

- **Entity Authentication + implicit key authentication + key confirmation:** YES.
- **Perfect Forward Secrecy:** NO: The keys are entirely determined by the exchanged quantities.

A similar protocol (only (3) changes) can be used for entity authentication.

Quick Review

Asymmetric Needham-Schroeder:

- $K := H(k_1, k_2)$ with encrypted exchanges
- Full authentication
- No PFS

Hybrid

Encrypted Key Exchange (EKE)

Hybrid protocol (symmetric + asymmetric) resistant to dictionary attacks.

Initialization: A and B share password p.

Protocol:

1. $A \to B : A, E_p(pub_A)$ (A generates key pair, sends public key encrypted)
2. $A \leftarrow B : E_p(E_{pub_A}(k))$ (B generates session key k, double encryption)
3. $A \to B : E_k(r_a)$ (Key confirmation)
4. $A \leftarrow B : E_k(r_a, r_b)$
5. $A \to B : E_k(r_b)$

Advantages:

- Robust even if password p is weak.
- Eve cannot guess without also "breaking" the asymmetric algorithm.

Properties:

- Entity authentication + implicit + confirmation
- Perfect Forward Secrecy **if** $pub_A/priv_A$ is regenerated each time
- No PFS if long-term keys

Original Text

Hybrid KTP: Encrypted Key Exchange (EKE)
This protocol uses **symmetric and asymmetric schemes** to minimize the risk of cryptanalysis via dictionary attacks inherent to symmetric systems.
(Init.): A and B share a symmetric secret p (password).

(1) $A \rightarrow B : A, E_p(pub_A)$; A generates a public/private key pair and sends the public part to B encrypted with p.

(2) $A \leftarrow B : E_p(E_{pub_A}(k))$; B generates a session key k and sends it encrypted.

(3) $A \rightarrow B : E_k(r_a)$; A generates a random number and sends it encrypted with k.

(4) $A \leftarrow B : E_k(r_a, r_b)$; B generates r_b and sends it with r_a encrypted with k.

(5) $A \rightarrow B : E_k(r_b)$; Confirmation from A. If r_b = OK => END.

(6) and (2) are responsible for **key transport**; (3) to (5) for **key confirmation**.

This protocol is **robust even if the password** p shared between A and B is of poor quality. Indeed, Eve cannot attempt to guess without also "breaking" the asymmetric algorithm.
Properties:

- **Entity Authentication + implicit key authentication + key confirmation:** YES.
- **Perfect Forward Secrecy:** YES if the $pub_A/priv_A$ pair is regenerated for each protocol instance. NO if $pub_A/priv_A$ is a long-term key.

Quick Review

EKE (Hybrid):

- Password + asymmetric crypto
- Robust even with weak password
- PFS if keys regenerated each time

Symmetric with Key Distribution Center (KDC)

Symmetric Needham-Schroeder

Protocol with **Key Distribution Center (KDC)**.

Initialization: A shares K_{AT} with T (KDC), B shares K_{BT} with T.

Protocol:

1. $A \to T : A, B, r_a$
2. $A \leftarrow T : E_{K_{AT}}(r_a, B, k_{AB}, E_{K_{BT}}(k_{AB}, A))$
3. $A \to B : E_{K_{BT}}(k_{AB}, A)$
4. $A \leftarrow B : E_{k_{AB}}(r_b)$
5. $A \to B : E_{k_{AB}}(r_b - 1)$

Properties:

- Entity authentication of A to B
- Entity authentication of B to A (A never saw r_b)
- Implicit key authentication
- Key confirmation (only B knows A possesses the key)
- Perfect Forward Secrecy

Vulnerabilities:

- **Replay attacks:** A can replay (3) without B's control.
- **Known-key attack:** If an old key k is compromised, an adversary can make B accept it.

Solutions:

- **Key confirmation and mutual entity authentication:**

  Replace 3. and 4. with:

  3. $A \to B : E_{k_{AB}}(r'_a), E_{K_{BT}}(k_{AB}, A)$
  4. $A \leftarrow B : E_{k_{AB}}(r'_a - 1, r_b)$

- **Exchange freshness:**

  Add timestamp in 3.: $E_{K_{BT}}(k_{AB}, A, t)$

Original Text

Symmetric KTP with Key Distribution Center – Symmetric Needham-Schroeder
(Notation): Let T be the **Key Distribution Center**.
(Init.): A and T share the symmetric key K_{AT}. B and T share K_{BT}.

(1) $A \to T : A, B, r_a$; A generates a random number r_a and sends it to T with the identities.

(2) $A \leftarrow T : E_{K_{AT}}(r_a, B, k_{AB}, E_{K_{BT}}(k_{AB}, A))$; T generates k_{AB} and sends it encrypted.

(3) $A \rightarrow B : E_{K_{BT}}(k_{AB}, A)$; A forwards the packet to B.

(4) $A \leftarrow B : E_{k_{AB}}(r_b)$; Confirmation from B using k_{AB} and a random number r_b.

(5) $A \rightarrow B : E_{k_{AB}}(r_b - 1)$; Confirmation from A.

Properties:

- **Entity Authentication:**
  - A to B: YES.
  - B to A: NO: A never saw r_b (it could be $E_{k'}(r'_b)$).
- **Implicit Key Authentication:** YES (keys are always protected by K_{AT} and K_{BT}). However, in case of a known-key attack, this no longer holds for B.
- **Key Confirmation:** Only B is assured that A possesses the key due to the flaw described in entity authentication.
- **Perfect Forward Secrecy:** NO. If either K_{AT} or K_{BT} is compromised, session keys k immediately become visible.

Solution to achieve key confirmation and mutual entity authentication:
Replace (3) and (4) with:
(3') $A \rightarrow B : E_{k_{AB}}(r'_a), E_{K_{BT}}(k_{AB}, A)$
(4') $A \leftarrow B : E_{k_{AB}}(r'_a - 1, r_b)$
Provided the r_i are carefully controlled by the participants.
However: Beware of **reflection attacks**!
Problem: A can replay (3) as many times as desired, without any control from B. This problem worsens if an old key k is compromised:
Vulnerable to known-key attack: If a previously used session key k is obtained by an adversary C, it can easily make B accept it by replaying (3) and computing the challenge sent by B in (5). In this case, the properties of entity authentication, implicit key authentication, and key confirmation of A to B are also compromised.
Solution: Add a **timestamp** in (3) attesting to the freshness of the exchanges:
(3'') $A \rightarrow B : E_{K_{BT}}(k_{AB}, A, t)$ (this is the solution adopted by **Kerberos**).

Quick Review

Symmetric Needham-Schroeder:

- KDC generates and distributes k_{AB}
- Vulnerable to replay and known-key attacks
- Solution: Add timestamp

- Basis for Kerberos

Kerberos

Authentication and key distribution protocol based on Needham-Schroeder with corrections.

Architecture:

- **Authentication Server (AS):** Issues tickets for TGS.
- **Ticket Granting Server (TGS):** Issues tickets for services.
- **Tickets:** Encrypted structures containing session keys.

Simplified Protocol:

Phase 1: TGT (Ticket Granting Ticket) Request

1. $A \rightarrow AS : A, TGS, r_a$
2. $A \leftarrow AS : E_{K_A}(k_{AT}, r_a), Ticket_{AT} := E_{K_T}(A, TGS, t_1, t_2, k_{AT})$

Phase 2: Ticket Request for Service B

3. $A \rightarrow TGS : Authenticator_{AT} := E_{k_{AT}}(A, t), Ticket_{AT}, B, r'_a$
4. $A \leftarrow TGS : E_{k_{AT}}(k_{AB}, r'_a), Ticket_{AB} := E_{K_B}(A, B, t_1, t_2, k_{AB})$

Phase 3: Authentication with B

5. $A \rightarrow B : Authenticator_{AB} := E_{k_{AB}}(A, t), Ticket_{AB}, r''_a, [request]$
6. $A \leftarrow B : E_{k_{AB}}(r''_a), [response]$

Properties:

- Entity authentication (all entities)
- Implicit key authentication
- Partial key confirmation (not between A and AS)
- Perfect Forward Secrecy

Vulnerabilities:

- **Password guessing attacks** on $E_{K_A}(k_{AT}, r_a)$ (Solution: pre-authentication)
- **Replay attacks** if r_a is poorly controlled
- Requires clock synchronization

Original Text

Symmetric KTP with Key Distribution Center – Kerberos
Kerberos is a protocol for **entity authentication** and **key distribution** within a user network.
Originally, Kerberos was designed as a replacement solution to address security issues (weak authentication, cleartext transactions, etc.) inherent to UNIX environments.
Kerberos was created at MIT as part of the ATHENA project.
The first three versions were unstable. **Version 4** achieved considerable success in both industrial and academic environments and remains predominant. **Version 5**, although safer and better structured, is more complex and less performant, which has slowed its deployment.
Kerberos also defines a mode of collaboration between domains belonging to distinct administrative authorities (the **realms**). This allows users from one domain to use resources from another domain "without leaving" the secure Kerberos environment.
For inter-realm transactions, symmetric cryptography constitutes a significant obstacle as it requires confidential channels for key pre-distribution.
Kerberos Version 5
(Notation): - A and B want to establish a secure transaction; in the Kerberos environment, this typically involves a **client** and a **server** providing services. - The Kerberos KDC is subdivided into two functional entities: the **Authentication Server (AS)** and the **Ticket Granting Server (TGS)**. Both access the password database. - The $r_a^{(n)}$ are random numbers, t is a timestamp, t_1 and t_2 indicate a validity time window.
(Initialization): A and B share a secret key with AS, namely: K_A and K_B (for clients, this is a OWF of the password). TGS also has a secret key K_T.

(1) $A \rightarrow AS : A, TGS, r_a$

(2) $A \leftarrow AS : E_{K_A}(k_{AT}, r_a), Ticket_{AT} := E_{K_T}(A, TGS, t_1, t_2, k_{AT})$; AS generates k_{AT}.

(3) $A \rightarrow TGS : Authenticator_{AT} := E_{k_{AT}}(A, t), Ticket_{AT}, B, r'_a$

(4) $A \leftarrow TGS : E_{k_{AT}}(k_{AB}, r'_a), Ticket_{AB} := E_{K_B}(A, B, t_1, t_2, k_{AB})$; TGS generates k_{AB}.

(5) $A \rightarrow B : Authenticator_{AB} := E_{k_{AB}}(A, t), Ticket_{AB}, r''_a, [request]$

(6) $A \leftarrow B : E_{k_{AB}}(r''_a), [response]$; [request] and [response] optionally encrypted with k_{AB}.

(7) • (2): TGT request.

(8) • (4): Ticket request for B.

(9) • (6): Authentication and key establishment between A and B.

Kerberos Characteristics
Properties:

- **Entity Authentication:** YES, for all involved entities.
- **Implicit Key Authentication:** YES: All generated keys are protected by keys shared between the AS and all participants.
- **Key Confirmation:**
  - Between A and AS: NO: AS has no proof that A possesses the key K_A.
  - Between A and TGS: YES for k_{AT} (redundant quantities encrypted with k_{AT} are exchanged between A and TGS); NO for k_{AB} (TGS has no proof from A).
  - Between A and B: YES: Exchange of redundant quantities encrypted with k_{AB}.
- **Perfect Forward Secrecy:** NO: All keys are explicitly transferred.

Problems:

- Initial keys (like K_A) depend (directly) on user-chosen passwords. This makes the protocol vulnerable to password theft or:
  - **Password guessing attacks:** $E_{K_A}(k_{AT}, r_a)$ in (2) helps crack A's password. **Solution:** Pre-authentication in (1): $E_{K_A}(t)$ with t = timestamp (optional in v5).
- The ticket validity window can lead to **replay attacks** if the $r_a^{(n)}$ are not properly controlled by the participants.
- **Clock synchronization** is necessary! This is not always easy in heterogeneous environments.

Quick Review

Kerberos:

- AS issues TGT, TGS issues service tickets
- Tickets contain session keys
- Authentication via authenticators
- Vulnerable: password guessing, replay
- Solution: pre-authentication, timestamps

SSL/TLS

SSL/TLS: Secure Socket Layer / Transport Layer Security

Protocol for securing communications between the transport (TCP) and application layers.

Provided Services:

- Confidentiality, integrity, flow authentication
- Server identification (client optional)

Algorithms Used:

- **Public-key cryptography** (RSA, DH, DSA): Key exchange
- **MACs:** Flow authentication
- **Symmetric cryptography** (DES, AES, IDEA): Flow encryption

Properties:

- Entity authentication (server + optional client via certificates)
- Implicit key authentication
- Key confirmation
- Perfect Forward Secrecy: Depends on the exchange protocol (DH → yes, RSA → no)

Remarks:

- TLS keys are derived by hashing from random values and the *pre_master_secret*.
- SSL/TLS is the de facto standard for web security (HTTPS).
- Trust relies on root certificates embedded in browsers.
- Major vulnerabilities stem from randomness, implementations, and hash functions.
- Notable attacks: renegotiation (2009), Heartbleed (2014).

SSL/TLS Architecture

Three Components:

1. **SSL Record Protocol:** Encapsulation above TCP (fragmentation + compression + encryption)
2. **SSL Handshake Protocol:** Authentication + parameter negotiation
3. **SSL State Machine:** Session and connection state variables

> Original Text
>
> **Secure Socket Layer (SSL) / Transport Level Security (TLS)**
> Located between the transport layer (TCP) and application layer protocols (not only

HTTP but also SMTP, FTP, etc.!).
It is a **meta key establishment protocol** highly configurable, allowing many modes of operation and negotiation options.
Provides **confidentiality**, **integrity**, **data flow authentication**, and **server identification** (and optionally client identification) services.
Uses the following algorithm families:

- **Public-key cryptography** (RSA, Diffie-Hellman, DSA, etc.) for symmetric key exchange.
- **MACs** for data flow authentication.
- **Symmetric cryptography** (DES, IDEA, AES, etc.) for data flow encryption.

The use of **CAs** to certify the association between entities and public keys is strongly recommended… but not mandatory!
Properties: - **Entity authentication** via certificates (server and optionally client). - **Implicit Key Authentication** and **Key Confirmation** are guaranteed. - **Perfect Forward Secrecy** depends on the protocol chosen for key exchange.
SSL/TLS Overview
SSL is a "mini-stack" of protocols with functionalities from the session, presentation, and application layers.
SSL consists of three fundamental blocks:

- **SSL record protocol** allowing encapsulation of higher-level protocols above TCP (fragmentation + compression + encryption).
- **SSL handshake protocol** responsible for participant authentication and encryption parameter negotiation.
- **SSL state machine**. Unlike HTTP, SSL is a **stateful** protocol; it therefore requires a set of variables determining the state of a session and a connection.

SSL Handshake Protocol

Diagram

Phase 1: Hello

- **Client Hello:** Version, random, session ID, accepted algorithms
- **Server Hello:** Version, random, session ID, selected algorithms
- **Server Certificate** (optional): Server certificate + CA path
- **Server Key Exchange** (optional): Server public key information
- **Certificate Request** (optional): Client certificate request

Phase 2: Client Authentication and Key Exchange

- **Client Certificate** (optional): Client certificate + CA path
- **Client Key Exchange:** Generates `pre_master_secret`, sends encrypted with server's public key
- **Certificate Verify** (optional): Explicit client certificate verification

Phase 3: Finalization

- **Finish** (client): First message protected with negotiated parameters
- **Finish** (server): Same for the server

Phase 4: Application

- Data protected with derived keys

SSL/TLS Key Generation

Cascading Derivation:

$$master\_secret = MD5(pre\_master\_secret + SHA('A' + pre\_master\_secret + ClientRandom + ServerRando$$
$$+ MD5(pre\_master\_secret + SHA('BB' + ...)) + ...$$

$$key\_block = MD5(master\_secret + SHA('A' + master\_secret + ServerRandom + ClientRandom)) + ...$$

Partition of the key_block:

- `client_write_MAC_secret[hash_size]`
- `server_write_MAC_secret[hash_size]`
- `client_write_key[key_material]`
- `server_write_key[key_material]`
- `client_write_IV[IV_size]`
- `server_write_IV[IV_size]`

Original Text

Simplified SSL/TLS Handshake Protocol
[Handshake diagram with 4 phases: Hello, Key Exchange, Finish, Application Data]
SSL/TLS: Key Generation

```
master_secret =
 MD5(pre_master_secret + SHA('A' + pre_master_secret +
 ClientHello.random + ServerHello.random)) +
 MD5(pre_master_secret + SHA('BB' + pre_master_secret +
 ClientHello.random + ServerHello.random)) +
 MD5(pre_master_secret + SHA('CCC' + pre_master_secret +
 ClientHello.random + ServerHello.random));

key_block =
 MD5(master_secret + SHA('A' + master_secret +
 ServerHello.random +
 ClientHello.random)) +
 MD5(master_secret + SHA('BB' + master_secret +
 ServerHello.random +
 ClientHello.random)) +
 MD5(master_secret + SHA('CCC' + master_secret +
 ServerHello.random +
 ClientHello.random)) + [...];

 until enough output has been generated. Then the key_block is
 partitioned as follows:

 client_write_MAC_secret[CipherSpec.hash_size]
 server_write_MAC_secret[CipherSpec.hash_size]
 client_write_key[CipherSpec.key_material]
 server_write_key[CipherSpec.key_material]
 client_write_IV[CipherSpec.IV_size] /* non-export ciphers */
 server_write_IV[CipherSpec.IV_size] /* non-export ciphers */
```

SSL/TLS: Final Remarks

- Secret keys are the result of applying hash functions (MD5, SHA) to the random numbers from the Hello records and the `pre_master_secret`.
- TLS/SSL has become the **de facto standard** for web security (the basis of **https**).
- SSL clients (Explorer, Firefox, Opera, Chrome, etc.) contain "hard-coded" certificates corresponding to a few root certification authorities (Verisign, Thawte, Microsoft, RSA, etc.) allowing verification of certificates presented by some servers, but SSL is designed to rely on a **global certification network** that currently does not exist.
- The most common **security flaws** in SSL concern key randomness generation as

well as the most common implementation defects: buffer overflows, SQL injection, etc. The weakness of hash functions (MD5, SHA) is also a risk factor.

- In November 2009, an attack was discovered allowing a Man-in-the-Middle to inject content (chosen plaintext) into an authentic flow following a **renegotiation** of parameters provided for in the protocol. This is a flaw in the protocol that required a patch in all implementations.
- The **Heartbleed** vulnerability based on a buffer overflow seriously disrupted the Internet community upon its discovery in April 2014.

Quick Review

SSL/TLS:

- Meta-protocol between TCP and application
- Handshake: negotiation + authentication
- Keys derived: `master_secret` → `key_block`
- HTTPS standard
- Flaws: randomness, Heartbleed, renegotiation

Final Remarks on KEPs

Before choosing a KEP:

1. **Define objectives:** Confidentiality, authentication, non-repudiation.
2. **Define security level:** Key confirmation, PFS, future secrecy.
3. **Establish constraints:** Users, machines, network, attackers.

Best Practices:

- Choose a proven and robust solution.
- Avoid inventing "from scratch."
- Verify that properties are satisfied.

Protocol Verification:

Two complementary approaches:

- **Practical Analysis:** "On paper" and "on machine"
  - Control random numbers (reflection attacks)

  - Redundancy of encrypted/signed quantities
  - Classic pitfalls

- **Formal Analysis:** Dedicated logics (BAN logic, etc.)

Original Text

Key Establishment Protocols: Final Remarks
Key establishment protocols are a **cornerstone** of any security solution. Before choosing (designing) a KEP, it is therefore essential to:

- **Define the objectives** (confidentiality, entity/data authentication, non-repudiation, etc.).
- **Define the desired security level** based on the studied properties (key confirmation, perfect forward secrecy, etc.).
- **Establish a list of constraints** related to the environment (users, machines, network, potential attackers, etc.).

Based on these criteria, we can:

- **Choose a proven and robust solution** (better than inventing one from scratch!).
- **Verify that the objectives are met** and the properties satisfied.

Protocol verification is a complex and delicate process; moreover, published solutions are not always correct. Two approaches are possible (and necessary):

- **Practical analysis.** Analyze protocol flaws "on paper" and "on machine," considering classic pitfalls: control of random numbers to avoid reflection attacks, redundancy of encrypted/signed quantities, etc.
- **Formal analysis** with logics specifically designed for this purpose (such as **BAN logic**).

Quick Review

KEP – Best Practices:

1. Define objectives and constraints
2. Choose a proven solution
3. Verify properties (practical + formal)
4. Avoid pitfalls: reflection, redundancy, randomness control