The Challenge of
Session-Key Distribution Protocols

Mihir Bellare
University of California at San Diego
In practice, bulk data encryption or authentication is typically based on possession of a shared key $K$—

But how are Alice and Bob to obtain the shared key?
Diffie-Hellman Secret-Key Exchange Protocol

**Alice**

\[ x \overset{R}{\leftarrow} Z_{p-1} \]
\[ X \leftarrow g^x \mod p \]

**Bob**

\[ y \overset{R}{\leftarrow} Z_{p-1} \]
\[ Y \leftarrow g^y \mod p \]

\[ Y^x \mod p = g^{xy} \mod p = X^y \mod p \]

**Publicly known:**

- \( p \) – Prime number
- \( g \) – Generator of \( Z_p^* \)
Diffie-Hellman Secret-Key Exchange Protocol
Does not provide Authenticity

Publicly known:
\( p \) – Prime number
\( g \) – Generator of \( Z_p^* \)

This protocol is only secure against passive (i.e., eavesdropping) adversaries.
The **Session-Key Distribution (SKD)** problem

**Premise:** Existence of authentic long-lived (LL) keys based on which authentication is possible.

**Goal:** Parties wish to interact so as to come into possession of an authentic shared key to be used in a subsequent communication session.

**Focus:** The design and analysis of SKD protocols

**De-coupling:** No assumptions should be made about how $K$ is used in the communication session.
The **Three-Party Model**

**LL-keys and trust assumptions:**
- There is a trusted authentication server $S$
- $A$ shares a key (password) $a$ with $S$
- $B$ shares a key (password) $b$ with $S$

**SKD protocol problem:** $A, B, S$ interact to get a session key $K$

**Example systems:** Needham-Schroeder, Kerberos

**Example protocol:**

$A \rightarrow S$: $A, B, N_A$

$S \rightarrow A$: $\{N_A, B, K, \{K, A\}_b\}_a$

$A \rightarrow B$: $\{K, A\}_b$

$B \rightarrow A$: $\{N_B\}_K$

$A \rightarrow B$: $\{N_B - 1\}_K$
The Two-Party, Asymmetric Model

LL-keys and trust assumptions:
- There is a public-key infrastructure
- \( A \) has \( pk_B \), the public key of \( B \), and its certificate
- \( B \) has \( pk_A \), the public key of \( A \), and its certificate

SKD protocol problem: \( A, B \) interact to get a session key \( K \)

Example systems: SSL, TLS

Example protocol:
\[
A \xrightarrow{g^x} B \\
\xleftarrow{g^y, \text{Sig}_B(A, g^x, g^y)} \text{Sig}_A(g^y) \\
K \xrightarrow{K, K = g^{xy}} K
\]
**The Two-Party, Symmetric Model**

**LL-keys and trust assumptions:** $A$ and $B$ share a key (password) $a$

**SKD protocol problem:** $A, B$ interact to get a session key $K$

**Example protocol:**

```
A

$g^x$

$g^y, \text{Mac}_a(B, A, g^x, g^y)$

Mac$_a(A, g^y)$

$K$

$K = g^{xy}$

B
```
Usage of Session-Key Distribution Protocols

Across technologies
- End to end Internet communication
- Smart Card authentication
- Wireless
- Hand-held devices
- :

Across Applications
- Electronic payments
- Electronic banking
- Access control
- :
- :
Security goals for SKD:

A continually rising bar

- Authentication (unilateral or mutual)
- Freshness and secrecy of session key
- Resistance to known-key attacks
- Forward secrecy of session key
- Resistance to dictionary attacks
- Anonymity of mobile users
- :.

Implementation constraints:

Many possible kinds

- Processor speed
- Memory
- Power
- Bandwidth
- :.
SKD protocols are easy to propose but hard to get right. The area is notorious for broken protocols. Attacks sometimes take years to discover, by which time the protocols are deployed.
More protocols for SKD are proposed every year than for all other cryptographic problems combined.

The bulk of proposed protocols are extraordinarily poor, typically designed by non-cryptographers and appearing in IEEE journals.

Attacks on these protocols fuel the production of more literature.
Two main approaches to the security analysis of SKD protocols

**Formal methods:** From the security community
- Initiated by Burrows, Abadi, Needham (BAN90)
- Often successful at finding bugs
- Some approaches can be automated
- Model primitives as ideal objects, but it is not clear what this means
- No rigorous proofs of security

**References:** Survey articles by Meadows 92, Rubin and Honeyman 93.

**Provable security:** From the cryptographic community.
- Initiated by Bellare, Rogaway (BeRo93)
- Based on computational complexity theory and the GM approach
- Definitions of security for both primitives and goals
- Helps find bugs
- Security of (secure) SKD protocols is proved
Provable-Security in SKD protocol analysis

**Definitional approaches:**
- Direct (BeRo93, BeRo95, BIJoMe97, BlMe97, BePoRo00)
- Simulation-based (BeCaKr98, Sh99, BoMaPa00)

**Trust models treated:**
- Two-party, symmetric (BeRo93)
- Two-party, asymmetric (BIJoMe97, BlMe97, BePoRo00)
- Three party (BeRo95, ShRu96)

**Security goals treated:**
- Authentication, freshness and secrecy of session keys, known key attacks (BeRo93, BeRo95, ...)
- Forward secrecy (BePoRo00)
- Dictionary attacks (HaKr98, BePoRo00, BoMaPa00)
Efficient, Provably-Secure SKD protocols

are available for the following problems:

- Two-party, symmetric SKD (BeRo93)
- Three party session key distribution (BeRo95)

and, most importantly in current climate, also for

Two-party, asymmetric SKD with forward secrecy and security against dictionary attack (BePoRo00, BoMaPa00)

**Note:** The analyses for the last use the random oracle model.
Where we are in SKD

**Successes:** Provable-security has been very successful, especially for treating the basic goals of SKD.

**Getting harder:** As more and more goals are added, the complexity of definitions and proofs increases.

Some directions for the future

**Coming together:** Try to combine the provable security and formal methods approaches (Mitchell et al, Abadi, Abadi-Rogaway)

**Take responsibility for your protocols:** The burden of proof is on the protocol proposer, NOT the attacker. Stop proposing un-analyzed protocols!
Plan for this talk

[ ] Introduction: Trust models, approaches, state of the art
[ ] Example protocols and security goals
[ ] How provable-security works
Needham-Schroeder 78

Three party model: $S$ is a trusted server; $A$ shares a key $a$ with $S$; $B$ shares a key $b$ with $S$.

Notation:

- $\{ X \}_{\text{key}} \overset{\text{def}}{=} \text{Encryption of } X \text{ under key}$
- $N_{\text{party}} = \text{Nonce chosen by party}$

Protocol:

- $A \rightarrow S$: $A, B, N_A$
- $S \rightarrow A$: $\{ N_A, B, K, \{ K, A \}_b \}_a$
- $A \rightarrow B$: $\{ K, A \}_b$
- $B \rightarrow A$: $\{ N_B \}_K$
- $A \rightarrow B$: $\{ N_B - 1 \}_K$

Distribution phase

Freshness / Replay check

Session key: The distributed session key $K$ is chosen by $S$. 
**Denning-Sacco 81: Known-key attack on NS**

If a session key $K$ is revealed to Adversary after the session is over then later, a compromised session can be created.

**Adversary** has a record of the full conversation—

$$
A \rightarrow S: \quad A, B, N_A \\
S \rightarrow A: \quad \{N_A, B, K, \{K, A\}_b\}_a \\
A \rightarrow B: \quad \{K, A\}_b \\
B \rightarrow A: \quad \{N_B\}_K \\
A \rightarrow B: \quad \{N_B - 1\}_K
$$

Now $K$ is leaked. So **Adversary** does the following—

$$
\text{Adversary} \rightarrow B: \quad \{K, A\}_b \\
B \rightarrow A \quad \text{Adversary}: \quad \{N_B\}_K \\
\text{Adversary} \rightarrow B: \quad \{N_B - 1\}_K
$$

Now $B$ might encrypt a message under $K$ and send it to $A$. But **Adversary** can read it.
Goal for secure session-key distribution

**Security against known-key attacks:** Adversary should not be able to obtain session key of one session even if it knows session keys of other sessions.

**Rationale:** No assumptions should be made about how a communication session uses its session key.
Confusion about primitives

Second flow of NS:

\[ S \rightarrow A: \{N_A, B, K, \{K, A\}_b\}_a \]

where \( \{ X \}_{\text{key}} \overset{\text{def}}{=} \text{“Encryption” of } X \text{ under key} \)

How do we instantiate \( \{ \cdot \}_{\text{key}} \)?

CBC encryption is not good enough because here encryption is used to provide authenticity.
The Modern cryptography view of primitives

Encryption: \( \text{Enc}_{\text{key}[1]}(\text{Data}) \)
Provides privacy
Instantiations: CBC, CTR, and other modes of operation

MAC: Data, \( \text{Mac}_{\text{key}[2]}(\text{Data}) \)
Provides authenticity
Instantiations: CBC-MAC, HMAC, UMAC

Key Separation: Encryption and MAC functions use different keys, but they can be derived from a single higher level key.

Second flow of NS, done right:

\[ S \rightarrow A: \quad C_A, \text{Mac}_{\text{a}[2]}(B, N_A, C_A), \quad C_B, \text{Mac}_{\text{b}[2]}(A, N_A, C_B) \]

where

\[ C_A = \text{Enc}_{\text{a}[1]}(K) \]
\[ C_B = \text{Enc}_{\text{b}[1]}(K) \]
Benefits of separating encryption and authentication

- Modular and flexible software design
- Easier to understand rationale for flows
- Facilitates security analysis
- Improves performance
Quality of secrecy of session key

Last two flows of NS:

\[ B \rightarrow A: \quad \{N_B\}_K \]
\[ A \rightarrow B: \quad \{N_B - 1\}_K \]

This reveals information about \( K \) which can result in the insecurity of a communication session using \( K \).
Semantic security of session key:  Adversary should be unable to compute any partial information about $K$.

Rationale:  No assumptions should be made about how a communication session uses its session key. Without semantic security, some usages will be insecure.

Remark:  This rules out “key confirmation”. But that can be done differently.
A provably-secure three-party SKD protocol (BeRo95)

\[
\begin{align*}
A \rightarrow B: & \quad R_A \\
B \rightarrow S: & \quad R_A, R_B \\
S \rightarrow A: & \quad Enc_{a[1]}(K), Mac_{a[2]}(A, B, R_A, C_A) \\
S \rightarrow B: & \quad Enc_{b[1]}(K), Mac_{b[2]}(A, B, R_B, C_B)
\end{align*}
\]

Easy to understand!

Provably provides security against known-key attacks and semantic security of session key assuming the encryption and MAC functions meet normal, well-understood security goals.
A provably-secure two-party, asymmetric SKD protocol

Adapted from BeRo93

Two-party, asymmetric model: $A$ has $pk_B$ and $B$ has $pk_A$

Provably provides security against known-key attacks and semantic security of session key assuming the public-key encryption and signature functions meet normal, well-understood security goals.
August 15: **Adversary** captures the following transcript:

\[
\text{A} \xrightarrow{R_A} \text{B} \\
\xleftarrow{R_B, \text{Enc}_A(K), \text{Sig}_B(A, R_A, R_B, C_A)} \\
\xrightarrow{\text{Sig}_A(R_B)} \\
\]

August 15: **Adversary** captures \( C = \text{Enc}_K(\text{Data}) \)

September 15: **Adversary** breaks into \( A \)'s computer and obtains its LL secret key \( \text{sk}_A \)

September 15: **Adversary** can obtain

\[
K = \text{Dec}_A(C_A) \quad \text{and} \quad \text{Data} = \text{Dec}_K(C).
\]
Forward secrecy: Adversary should be unable to compute (any partial information about) $K$, even if it obtains the LL secret key of either party after the execution of the protocol distributing $K$ has completed.

Rationale: In practice, computer systems are less secure than protocols, and forward secrecy mitigates the damage caused by key compromise due to break-ins.
SKD Protocol providing forward secrecy

Variant of the STS protocol of DiVaWi92

\[ A \]
\[
\begin{align*}
x & \overset{R}{\leftarrow} Z_{p-1} \\
X & \leftarrow g^x \mod p
\end{align*}
\]

\[ B \]
\[
\begin{align*}
y & \overset{R}{\leftarrow} Z_{p-1} \\
Y & \leftarrow g^y \mod p
\end{align*}
\]

\[ Y, \text{Sig}_B(A, X, Y) \]
\[ \text{Sig}_A(Y) \]

\[ K = \text{Hash}(g^{xy} \mod p) \]
Raising the bar further: Dictionary attacks

Client $A$ has a short password $pwd \in \text{Dictionary}$ which is known to server $B$

Adversary captures the following transcript:

$$
\begin{align*}
A & \quad X \\
X & \rightarrow B \\
\text{RealMac} & \\
Y, \text{Mac}_{pwd}(B, A, X, Y) & \\
\text{Mac}_{pwd}(A, Y) & \leftarrow
\end{align*}
$$

Dictionary attack:

For all $\text{CandidatePwd} \in \text{Dictionary}$ do

$$
\begin{align*}
\text{CandidateMac} & \leftarrow \text{Mac}_{\text{CandidatePwd}}(B, A, X, Y) \\
\text{If } \text{CandidateMac} = \text{RealMac} & \text{ then return } \text{CandidatePwd}
\end{align*}
$$
Another new goal for secure session-key distribution

**Security against dictionary attack:** The number of candidate passwords that can be eliminated by off-line search is at most the number of protocol transcripts possessed by the adversary.

**Rationale:** This is the best possible security in the face of the existing practice of using poorly chosen passwords.

**Lots of protocols:**
- Bellovin and Merritt 92, 93 — EKE
- Ja96, Ja97, Lu97, MaSw99, Wu98, HaKr98, BePoRo00, BoMaPa00
This protocol provably provides security against known-key attacks, semantic security of session key, forward secrecy and security against dictionary attacks in the random oracle model, assuming the Diffie-Hellman problem is hard in $G$.

**Publicly known:**
- Prime number $q$; Generator $g$ of a group $G$ of order $q$;
- Ideal cipher $E$; Random oracle hash function $\text{Hash}$.

**This protocol**
- $K = \text{Hash}(A, B, X, Y, g^{xy})$;
- $x \overset{R}{\leftarrow} Z_q$; $X \leftarrow g^x$;
- $A, E_{\text{pwd}}(X)$;
- $B$
- $y \overset{R}{\leftarrow} Z_q$; $Y\leftarrow g^y$;
- $E_{\text{pwd}}(Y)$
Plan for this talk

[ x ] Introduction: Trust models, approaches, state of the art
[ x ] Example protocols and security goals
[ ] How provable-security works
Two kinds of attacks on SKD protocols:

- Arising from weaknesses in the underlying primitives
  Example: break block cipher; MAC forgery

- Arising from flaws in SKD protocol design
  Example: known-key attack on NS

Goal of provable-security: Prove the absence of attacks arising from protocol flaws.

Assume: Absence of attacks on the underlying primitives.
Methodology of provable security

Model – What the adversary can do
Definition – When is the adversary successful
Proof – If the adversary succeeds we can break the underlying primitives

Based on clear, standard and well-understood cryptographic properties of the underlying primitives.
Sources of definitional ideas in SKD

Basic model and definitional ideas from Bellare-Rogaway 93–
- Instances of parties
- Adversary communicating with its instance oracles via queries
- Adversary in control of all communication
- Freshness via partnering
- Semantic security of session keys

Some variations following Bellare-Pointcheval-Rogaway 00–
- Partnering via session ids
- Fixing some oversights: timing of the test query
- Extensions to forward secrecy and dictionary attacks
We will treat **two-party, asymmetric SKD**

---

**Canonical Protocol**

```
A

// sid, K
// K, sid

B
```  

**Example Protocol**

```
A

// sid = (R_A, R_B)

B

// R_B, Enc_A(K), Sig_B(A, R_A, R_B, C_A)

// Sig_A(R_B)
```  

---

Fix $\Pi$ — Protocol description function / syntax which specifies all the software: The LL-key generation algorithm; The underlying primitives; The instructions for computing and verifying flows and for computing $K, sid$.

---

The question: Is $\Pi$ secure?
Instances of parties

Instances of $A$ – $A_1, A_2, A_3, \ldots$ – Each holds $sk_A, pk_B$
Instances of $B$ – $B_1, B_2, B_3, \ldots$ – Each holds $sk_B, pk_A$

Instances run SKD protocol $\Pi$ and are the endpoints of sessions
Many instances may be active concurrently

**Session IDs pair-up instances:** If $A_i$ has $sid$ then
- $A_j$ cannot have the same session ID ($j \neq i$)
- There is at most one $B_j$ —called $A_i$’s partner— with this $sid$
- $A_i, B_j$ have the same session key $K$
The experiment of executing $\Pi$ with Adversary

Fix protocol description $\Pi$ and adversary algorithm Adversary.

- Run key generation algorithm to get public/secret keys for $A, B$.
- Give $sk_A, pk_B$ to $A_1, A_2, \ldots$
- Give $sk_B, pk_A$ to $B_1, B_2, \ldots$
- Give $pk_A, pk_B$ to Adversary

Adversary interacts with instances via oracle queries.

**Send queries:** Adversary controls communication between instances

- send($A_i, R$) — String $R$, chosen by Adversary, is sent to $A_i$
- $A_i$ runs $\Pi$ to compute the reply and returns it to Adversary
  
  **Example:** $R_B, \underbrace{Enc_A(K)}_{C_A}, \underbrace{\text{Sig}_B(A, R_A, R_B, C_A)}$

- 

When an instance accepts, the session id is returned to the Adversary.
The experiment of executing $\Pi$ with Adversary, continued

Reveal queries: To model known-key attacks
- reveal($A_i$) – Adversary asks for $A_i$’s session key
- If $A_i$ has accepted then its session key is returned to Adversary

Test queries: To model semantic-security of session key
- test($A_i$) – Adversary claims it can predict $A_i$’s session key
- Flip a coin and return TestKey to Adversary where
  - If coin = 0 then TestKey = $K$
  - If coin = 1 then TestKey is a random string
- Adversary outputs a guess as to value of coin

The test query is allowed only if $A_i$ is fresh, meaning Adversary does not already know its session key, meaning
- No reveal($A_i$) query was made
- No reveal($B_j$) query was made where $B_j$ is $A_i$’s partner
**Definitions of security** for SKD protocol $\Pi$

1. **Authenticity**: Adversary should be unable to disrupt the session-id based partnering.

   The probability of the following event is negligible: There exist distinct $i, j$ such that
   - Either $A_i$ and $A_j$ output the same session ID
   - or $B_i$ and $B_j$ output the same session ID

2. **Semantic security of session key**: Adversary should be unable to distinguish the real session key of a fresh instance from a random string.

   The excess over $1/2$ of the probability that Adversary returns the correct value of coin in its answer to its test query is negligible.

These conditions must hold for all probabilistic, polynomial time Adversary algorithms.
How the model helps find bugs

**New protocol:** Replaced signature on last flow of previous protocol with a MAC to improve performance.

This protocol is insecure.

**Adversary** can attack the semantic security of the session key via—
- Obtain a transcript between $A_1, B_1$ via send queries
- Query $\text{test}(A_1)$ to get back a test key $\text{TestKey}$
- If $\text{Mac}_{\text{TestKey}}(A, R_B) = \text{RealMac}$ then $\text{TestKey} = K$ else $\text{TestKey}$ is random
Theorem: The protocol \( \Pi \) above satisfies authenticity and semantic security of session keys assuming the signature scheme meets the notion of unforgeability under chosen-message attack and the encryption scheme meets the notion of indistinguishability under chosen plaintext attack.
Reductionist approach to prove the security theorem

Given Adversary violating one of the security properties of the SKD protocol, construct a forger Forger and an eavesdropper Eve such that
- Either Forger breaks the signature scheme, or
- Eve breaks the encryption scheme

The new adversary algorithms will run Adversary, simulate the instance oracles, and finally use the success of Adversary to win their own games.
A flexible framework

To model extra security goals, we enhance adversary capabilities by adding new kinds of oracle queries, and then make new definitions.

**Corrupt queries:** To model forward secrecy

- \( \text{corrupt}(A) \) – Adversary corrupts party \( A \)
- Adversary gets back \( sk_A \)

Forward secrecy is captured by semantic security of session keys in face of extended adversary capabilities and new notion of freshness.

**Execute queries:** To model dictionary attacks

- \( \text{execute}(A_i, B_j) \) – Adversary requests transcript
- Adversary gets back a transcript of a full interaction

Security against dictionary attacks is captured via the concrete security statement of the theorem.
The provable-security approach has been successful in delivering improved security guarantees for session-key distribution protocols.

We now have provably-secure and efficient SKD protocols achieving not just basic goals like semantic security of session keys but also forward secrecy and security against dictionary attacks.

Many challenges remain in the design and analysis of session-key distribution protocols.