Outline

1. Role of Security Protocols
2. Aspects of Security and Cryptographic Algorithms
   - Security Properties
   - Attacker Models
   - Keys
   - Symmetric and Asymmetric Systems
3. Security Protocols
   - Notation and Examples
   - Vulnerabilities and Attacks
4. Formal Approaches
   - Authentication Logics
   - State Enumeration
   - Inductive Method
5. Summary
Outline

1. Role of Security Protocols
2. Aspects of Security and Cryptographic Algorithms
   - Security Properties
   - Attacker Models
   - Keys
   - Symmetric and Asymmetric Systems
3. Security Protocols
   - Notation and Examples
   - Vulnerabilities and Attacks
4. Formal Approaches
   - Authentication Logics
   - State Enumeration
   - Inductive Method
5. Summary
Outline

1. Role of Security Protocols
2. Aspects of Security and Cryptographic Algorithms
   - Security Properties
   - Attacker Models
   - Keys
   - Symmetric and Asymmetric Systems
3. Security Protocols
   - Notation and Examples
   - Vulnerabilities and Attacks
4. Formal Approaches
   - Authentication Logics
   - State Enumeration
   - Inductive Method
5. Summary
Outline

1. Role of Security Protocols
2. Aspects of Security and Cryptographic Algorithms
   - Security Properties
   - Attacker Models
   - Keys
   - Symmetric and Asymmetric Systems
3. Security Protocols
   - Notation and Examples
   - Vulnerabilities and Attacks
4. Formal Approaches
   - Authentication Logics
   - State Enumeration
   - Inductive Method
5. Summary
Role of Security Protocols

- Critical element of the infrastructure of a distributed system.
- Meant to provide a secure communication over an insecure network.
- Simple, short and easy to express.
- Extremely subtle and hard to evaluate.
- “Three-line programs that people still manage to get wrong” (Roger Needham).
- Excellent candidates for rigorous formal analysis.
Example: Exchanging Messages.

Monica and Bill run a security protocol: a previously agreed message handshake.

Hi Bill. It’s Monica!

So how’re you, Monica?

Fine, you?
Example: A Middle-Person Attack

If not designed correctly, security protocols are vulnerable to attacks.
Brief overview of formal analysis and validation methods for security protocols.
The First Question

What exactly do we want to protect when constructing a “security system”? 
Secrecy (Concealment, Confidentiality)

- Different possible meanings.
- Appropriate meaning decided by the designer of an application.
- The strongest interpretation: Intruders are not able to learn anything about any communication between two participants of a system.
- The weakest interpretation: Intruders are not able to read only the actual content of the messages.
Secrecy Interpretations: Pros and Cons

The strongest interpretation:

Pros: Is approximated closely by today’s cryptographic tools (encryption, digital signatures, dummy traffic, huge calculation overhead on the recipients).

Cons: In practice neither efficient nor necessary.

The weakest interpretation:

Pros: No useless overhead.

Cons: Intruders can know something about length of the message and log traffic analysis.
Two interpretations:

- **Strong authentication**: if a recipient $R$ receives a message claiming to be from a sender $S$ then $S$ has sent exactly this message to $R$.

- **Weak authentication**: if a recipient $R$ receives a message claiming to be from a sender $S$ then either $S$ has sent exactly this message to $R$ or $R$ notices this is not the case.
Authentication

- Digital signature systems: Authentication systems with non-repudiation property.
- Non-repudiation: A recipient is not only confident that the message is authentic (sent by $S$ and unmodified) but can also prove this fact to a third party.
Other Security Properties

- Anonymity
- Fairness
- Availability
The Second Question

- We *know* what to protect when constructing a “security system”.
- We *want to know* who to protect it from.
Relative Security

- Every kind of security needs an ultimately trusted physical support.
- Nothing can be protected from an almighty attacker.
- No absolute security.
- Security against specific types of attackers.
- Attacker specification can only be estimated.
In every distributed system there must be something that distinguishes the legitimate recipient from all other participants.

In cryptography: knowledge of a specific secret: key.
Key Generation

- Based on a truly random number.
- Very big key space to prevent identical keys and right guesses.
- Verification of relationship key $\leftarrow\rightarrow$ owner

The whole system is at most as good and trustworthy as the initial key generation.
Symmetric Cryptography

- Equal keys for encryption/decryption, signing/testing.
- Several thousand years old.
- Examples: Vernam chiffre (one time pad), DES (Data Encryption Standard), AES (Advanced Encryption Standard).
Symmetric Systems. Notational Conventions

\[ \mathcal{X} \quad \text{set of all plaintext messages} \]
\[ \mathcal{C} \quad \text{set of all ciphertexts} \]
\[ \mathcal{S} \quad \text{set of all signatures} \]
\[ \mathcal{K} \quad \text{set of all symmetric keys} \]
\[ k_{AB} \in \mathcal{K} \quad \text{specific key belonging to A and B} \]
Symmetric Secrecy.

\[ \text{encrypt} : \mathcal{X} \times \mathcal{K} \rightarrow \mathcal{C} \]
\[ \text{decrypt} : \mathcal{C} \times \mathcal{K} \rightarrow \mathcal{X} \]

\[ \forall k \in \mathcal{K}, x \in \mathcal{X}. \text{decrypt}(\text{encrypt}(x, k), k) = x \]

Sending an encrypted message from \( A \) to \( B \):

- Encryption: \( A \) chooses a message \( x \in \mathcal{X} \) and calculates \( c = \text{encrypt}(x, k_{AB}) \).
- Transfer: \( c \) is now sent to the recipient (and possibly to observers and attackers).
- Decryption: \( B \) calculates \( x = \text{decrypt}(c, k_{AB}) \).
Symmetric Authentication.

\[ \text{sign} : \mathcal{X} \times \mathcal{K} \rightarrow \mathcal{S} \]

Sending a signed message from \( A \) to \( B \):

- **Signing**: \( A \) chooses a message \( x \in \mathcal{X} \) and calculates \( s = \text{sign}(x, k_{AB}) \).
- **Transfer**: \( x; s \) is now sent to the recipient (and possibly to attackers)
- **Receiving**: \( B \) receives a message \( x' \); \( s' \) (either the original or modified by attackers)
- **Rest**: \( B \) calculates \( s'' = \text{sign}(x', k_{AB}) \); if \( s'' = s' \), the message is valid.
Symmetric Key Distribution.

- To use algorithms, participants have to agree on a common key.
- Easy if they can meet.
- Harder if they can not meet: Trusted third party needed.
- Exchange must be secret and authentic.

Solved by the Needham-Schroeder-Secret-Key (NSSK) protocol.
Asymmetric Cryptography

- Different keys for encryption/decryption, signing/testing.
- First paper: 1976 (Diffie and Hellmann)—key exchange.
- Based on one-way function.
- Used conjectures: factorization, discrete logarithm.
- Breakthrough of “cryptography for the masses”: PGP (Pretty Good Privacy), GPG (GNU Privacy Guard).
Asymmetric Systems. Notational Conventions.

- $\mathcal{X}$: set of all plaintext messages
- $C$: set of all ciphertexts
- $S$: set of all signatures
- $\mathcal{PUB}$: set of all public keys
- $\mathcal{SEC}$: set of all secret keys
- $pub_A \in \mathcal{PUB}$: specific public key of $A$
- $sec_A \in \mathcal{SEC}$: specific secret key of $A$
Asymmetric Secrecy.

\[
\text{encrypt} : \mathcal{X} \times \mathcal{PUB} \rightarrow \mathcal{C} \\
\text{decrypt} : \mathcal{C} \times \mathcal{SEC} \rightarrow \mathcal{X}
\]

\[\forall k \in \mathcal{K}. \text{decrypt} (\text{encrypt}(x, pub_A), sec_A) = x\]

Sending an encrypted message from A to B:

- Encryption: A chooses a message \(x \in \mathcal{X}\) and calculates \(c = \text{encrypt}(x, pub_B)\).
- Transfer: \(c\) is now sent to the recipient (and possibly to observers and attackers).
- Decryption: B calculates \(x = \text{decrypt}(c, sec_B)\).
Asymmetric Authentication.

\[
\begin{align*}
\text{sign} : \mathcal{X} \times \text{SEC} & \longrightarrow S \\
\text{test} : \mathcal{X} \times S \times \text{PUB} & \longrightarrow \{ \text{correct}, \text{wrong} \}
\end{align*}
\]

Sending a signed message from A to B:

- **Signing**: A chooses a message \( x \in \mathcal{X} \) and calculates \( s = \text{sign}(x, \text{sec}_A) \).
- **Transfer**: \( x; s \) is now sent to the recipient (and possibly to attackers)
- **Receiving**: B receives a message \( x' \); \( s' \) (either the original or modified by attackers)
- **Rest**: B now checks if \( \text{test}(x', s', \text{pub}_A) = \text{correct} \).
Protocols

- Protocol: a prescribed sequence of interactions between entities designed to achieve a certain goal and end.
- Security protocols provide security properties to distributed systems.
Security Protocol Hierarchy

- Cryptographic algorithms: Encrypting and decrypting data given suitable keys.
- Cryptographic protocols: Rules for exchanging information between agents.
- Key distribution protocols: Rules for maintaining and distributing keys to agents.
- Authentication protocol: Verifying the identities of agents.
- Certification: Allows trustworthy use of public keys.
- Non-repudiation protocols: Used when the agents do not trust each other.
Notation

Message

\[ a \rightarrow b : data \]

*data* consists of:

- **atoms**: names, variables, literal constants.
- **nonces**: \( n_A \) unpredictable, freshly generated unique number.
- **encryption**: \( data_k \): encryption of \( data \) with the key \( k \).
- **authentication**: \( sign_k(data) \): signature of \( data \) using the key \( k \).
- **concatenation**: \( data_1 . data_2 \)
Example: Challenge-Respond Protocol

**Purpose:** Verify that two parties $A$ and $B$ share a common secret key $k$ without revealing it.

1. $A \rightarrow B$: $n_A$
2. $B \rightarrow A$: $n_{Ak} \cdot n_B$
3. $A \rightarrow B$: $n_{Bk}$
Example: Needham-Schroeder Secret Key Protocol

**Purpose:** Establish a common secret key between $A$ and $B$ using only symmetric cryptography and a trusted third party $S$ (server).

**Preliminary:** participants have pairwise distinct keys with $S$.

1. $A \rightarrow S$: $A.B.n_A$
2. $S \rightarrow A$: $\{n_A.B.k_{AB}.\{k_{AB}.A\}_S\}_S^A$
3. $A \rightarrow B$: $\{k_{AB}.A\}_S^B$
4. $B \rightarrow A$: $n_{Bk_{AB}}$
5. $A \rightarrow B$: $\{n_B - 1\}k_{AB}$

(There exists an attack on this protocol.)
Example: Station-To-Station Protocol

**Purpose:** Establish a common secret key between $A$ and $B$ without trusted third party

1. $A \rightarrow B: a^x$
2. $B \rightarrow A: a^y . \{\text{sign}_B(a^y.a^x)\}_k$ \hspace{1cm} (k = a^{xy})
3. $A \rightarrow B: \{\text{sign}_A(a^y.a^x)\}_k$
List of Some Well-Known Protocols

1. Needham-Schroeder (flawed and corrected).
2. Netscape’s Secure Sockets Layer, SSL (flawed).
3. Transport Layer Security, TLS.
4. Otway-Rees shared-key.
5. Yahalom.
6. Wide-Mouthed Frog
7. Secure Multipurpose Internet Mail Exchange, S/MIME.
8. CCITT’s X.509

Protocol repository: http://www.lsv.ens-cachan.fr/spore/
Most Common Forms of Attacks

1. **Eavesdropping**: The intruder reads messages passing between sender and receiver, without their knowledge.

2. **Blocking**: The intruder intercepts a message and prevents it reaching the intended recipient.

3. **Forging**: The intruder sends messages to other agents which purport to be from someone else.
Forcing Example: Replay Attack

1. Attacker monitors a (possibly partial) run of a protocol and later replays some messages.
2. It can happen if the protocol does not have any mechanism for distinguishing between separate runs or cannot determine the freshness of messages.
3. **Example**: military ship that gets encrypted commands from base.
4. **Solutions**: Nonces, run identifiers, timestamps, indeterministic encryption.
Forging Example: Mirror Attack

Other participant is made to answer his own questions.

Vulnerability on Challenge-Response ($A$ does not know $k$).

If the system permits parallel login procedures, an attacker $A$ can fool the server $S$ with a second “dummy” session $A'$:

1. $A \rightarrow S$: $n_A$ (first login request)
2. $S \rightarrow A$: $n_{A_k} \cdot n_S$ (attacker cannot encrypt $n_S$ by now)
3. $A' \rightarrow S$: $n_S$ (second “dummy” login request)
4. $S \rightarrow A'$: $n_{S_k} \cdot n'_S$ (attacker got encrypted $n_S$ and abandons this session)
5. $A \rightarrow S$: $n_{S_k}$ (taken from the last answer)
Algebraic Attacks

- The attacks above are examples of “logical” attacks.
- “Algebraic” attacks: To break the underlying cryptography algorithm.
- “Algebraic” attacks are not considered in this talk.
Some Formal Approaches

1. Authentication logics: Modal “belief” Logic (BAN logic).
2. State enumeration: model checking.
3. Inductive method.
4. Narrowing/Rewriting (not in this talk).
5. Protocol refinement (not in this talk).

Verification by analysis: 1–4.
Verification by construction: 5.
Modal “belief” Logic

- BAN Logic: Burrows, Abadi, Needham.
- Models agent beliefs:
  - Nonce $N$ is fresh;
  - Key $K_{AB}$ is good;
  - Agent $S$ can be trusted.
Some Constructs of “belief” Logic

Examples of formulae:

\[ \alpha \models \phi \quad \text{agent } \alpha \text{ believes formula } \phi \]

\[ \alpha \triangleleft \phi \quad \text{agent } \alpha \text{ sees } \phi \]

\[ \alpha \triangleright \phi \quad \text{agent } \alpha \text{ once said } \phi \]

\[ \alpha \leftrightarrow k \beta \quad \alpha \text{ and } \beta \text{ may use shared key } k \text{ to communicate} \]

\[ \{ \phi \}_k \quad \text{formula } \phi \text{ is encoded with the key } k \]
Some Constructs of “belief” Logic

An example of inference rule:

$$\alpha \equiv \alpha \leftarrow k \rightarrow \beta, \quad \alpha \triangleleft \{\phi\}_k$$

$$\alpha \equiv \beta \sim \phi$$

- **Antecedent**: $\alpha$ believes that $k$ is a key shared with agent $\beta$ and $\alpha$ has received (sees) formula $\phi$ encrypted with key $k$.
- **Consequent**: $\alpha$ believes that agent $\beta$ has previously disclosed the (plaintext) formula $\phi$. 
First two steps of Otway-Rees Protocol:

1. $A \rightarrow B : N_X.A.B.\{N_A.N_X.A.B\}_{k_{AS}}$
2. $B \rightarrow S : N_X.A.B.\{N_A.N_X.A.B\}_{k_{AS}}.\{N_B.N_X.A.B\}_{k_{BS}}$

Representation in “belief” logic:

1. $A \rightarrow B : \{N_A, N_C\}_{k_{AS}}$
2. $B \rightarrow S : \{N_A, N_C\}_{k_{AS}}, \{N_B, N_C\}_{k_{BS}}$

Plaintext part is omitted.

$N_C$ replaces the compound datum $N_X.A.B.$
Analyzing Protocols in “belief” Logic. Example

Implicit assumption: S initially knows that the key $K_{AS}$ is shared between A and S:

$$S \equiv A^{k_{AS}} S.$$  

After the second step of Otway-Rees Protocol S sees the encrypted message components from B:

$$S \triangleleft \{ N_A, N_C \}^{K_{AS}}, \{ N_B, N_C \}^{K_{BS}}.$$  

From these two formulae, the inference rule deduces

$$S \equiv A \sim (N_A, N_C).$$
“Belief” Logic: Pros and Cons

**Pros:** Allows short, abstract proofs.

**Cons:** Misses many flaws, e.g., the translation of protocols into an idealized form has led to erroneous proofs that overlook the role of plaintext information in ensuring security.
Model Checking

- Cheap alternative to formal proof.
- An automatic tool which explores the state space of a model in an attempt to find illegal states.
- For models with a small state space the results are equivalent to a formal proof (exhaustive search).
- Even an incomplete search may succeed in finding a “bad” state (often more efficient than via formal proof).
Model Checking for Security Protocols

- Suitable notations: Process algebras, Petri nets, ...
- Built-in features for event ordering and communication.
- Successfully used in analysis of communication protocols.
- However, security also relies on the contents of the messages transmitted.
- Model checking languages lack data structure support.
- Notation has to be extended.
- Language used in examples in this talk: Communicating Sequential Processes (CSP) process algebra extended with a recursive data structure.
Model Checking for Security Protocols. Example

Fragment of reduced Needham-Schroeder Public Key protocol:

1. \( A \rightarrow B : \{N_A, A\}_{K_B} \)
2. \( B \rightarrow A : \{N_A, N_B\}_{K_A} \)

Encoding in CSP:

\[
AGENT = l_{running}.A.B \rightarrow \text{comm!Msg1}.A.B.\text{Encrypt.key}(B).N_A.A \rightarrow \text{comm.Msg2}.B.A.\text{Encrypt.key}(A)?N_A.N_B \rightarrow \ldots
\]

\[
l_{commit}.A.B \rightarrow \ldots
\]
Model Checking for Security Protocols. Example

Part of an intruder process in CSP:

\[
\text{INTRUDER} =
\]

\[
\ldots
\]

\[
\text{intercept.Msg1?A.B.Encrypt.K.N.A'} \rightarrow \ldots
\]

\[
\square\text{intercept.Msg2?B.A.Encrypt.K.N.N'} \rightarrow \ldots
\]

\[
\square\text{intercept.Msg3?A.B.Encrypt.K.N} \rightarrow \ldots
\]

\[
\ldots
\]
SPI-Calculus. Alternative to CSP

- CSP: *Process algebra extended* by features specifically for protocol verification.

- Spi-calculus: *Process algebra designed* specifically for protocol verification (Abadi and Gordon).

- Spi-calculus is based on a powerful process algebra, $\pi$-calculus, and extends it with key-encryption primitives.
Larry Paulson, using the Isabelle prover.

- Inductive method borrows ideas from “belief” logic and model checking:
- From “belief” logic: The idea of deriving guarantees from each message.
- From model checking: A concrete notion of event (A sending $X$ to $B$).
Protocols are formalized as the set of all possible traces, which are lists of events.

An agent may extend a trace in any way permitted by the protocol, given what he can see in the current trace.

Agents do not know the true sender of a message and may forward items that they cannot read.

One agent is an active attacker.

Properties are proved by induction on traces by Isabelle.
Formalizing Protocols

- The approach is oriented around proving guarantees.
- Their absence can indicate possible attacks.
- An attack on a variant of the Otway-Rees protocol was discovered (The protocol was believed to be safe, according to the “belief” logic).

Formal Methods for Security Protocols

Temur Kutsia
Modeling Protocol with Inductive Method. Example

Fragment of Needham-Schroeder Public Key protocol:

1. \( A \rightarrow B : \{N_A, A\}_{K_B} \)
2. \( B \rightarrow A : \{N_A, N_B\}_{K_A} \)

Encoding:

1. \( NS^1 \ (t_1 \in tr \land A \neq B \land nonce(N_A) \notin used(t_1)) \Rightarrow (says(A, B, crypt(pubk(B), \langle nonce(N_A), ag(A)\rangle))\#t_1) \in tr \)
2. \( NS^2 \ (t_2 \in tr \land A \neq B \land nonce(N_B) \notin used(t_2) \land says(X, B, crypt(pubk(B), \langle nonce(N_A), ag(A)\rangle)) \in set(t_2)) \Rightarrow (says(B, A, crypt(pubk(A), \langle nonce(N_A), nonce(N_B)\rangle))\#t_2) \in tr \)
Modeling Protocol with Inductive Method. Example

Powerful feature: Predicts unforeseen attacks by synthesizing all possible fraudulent messages that could be generated using the observable information in the current trace.

Using functions:

- **spies**\((t)\): All the messages an intruder can see in a trace \(t\).
- **analz**\((H)\): All the plaintext data that can be extracted from a set of messages \(H\), including plaintext components, and the contents of encrypted components for which a corresponding key can be found in \(H\).
- **synth**\((H)\): All the messages an intruder can build from a set \(H\) of message components.
Modeling an attack:

\[ \text{Fake}\ (t \in tr \land Y \neq I \land X \in \text{synth}(\text{analz}(\text{spies}(t)))) \Rightarrow \]
\[ (\text{says}(I, Y, X) \# t) \in tr \]

If there is a message \( X \) that intruder \( I \) can compose from the observable data in trace \( t \),

then the intruder will send such a message to some other agent \( Y \).
Facts that Can Be Proved by the Induction Method

- Secret keys are never lost.
- Nonces uniquely identify their message of origin.
- Nonces stay secret (under certain conditions!)

Proved by induction, simplification and classical reasoning.
Summary

- Security protocols have posed a significant challenge to formal methods.
- Wide range of formal methods have been used for security protocols.
- There is no clear “winner” among the methods used.
- Inductive method and spi-calculus appear to be the most elegant products.
Reading

C.J. Fridge.

L.C. Paulson.
The Inductive Approach to Verifying Cryptographic Protocols.

M. Pitt.
Modeling and Verification of Security Protocols.
Advanced seminar paper, Dresden University of Technology, Germany, November 2002.
Appendix

Factorization and Discrete Logarithm

Factorization: It is easy to take two different large prime numbers $p$ and $q$ and calculate their product $n = p \cdot q$. But no efficient algorithm is known to calculate the two prime factors of $n$.

Discrete logarithm: Taking an exponent is easy in finite fields: Given two numbers $a$, $x$, and a prime $p$ it is easy to calculate $y = a^x \mod p$. But on the other hand, there is no known algorithm that, given $y$, $a$, and $p$, can efficiently determine $x$. 