# Verification of Cryptographic Protocols

**Bart Jacobs**

`bart@cs.kun.nl` `http://www.cs.kun.nl/~bart`

Department of Computer Science
University of Nijmegen



## Outline





## I. Computer Security

- Challenging and exciting field of computer science, because new, complex problems arise continuously.
- Is relevant to all of the major topics in computing: operating systems, networks, data bases.

> Cryptographic coding is the most powerful tool in providing computer security.



## Security Protocols

Protocols are orderly sequences of steps that two or more parties take in order to accomplish a task, for mutual benefit.

Security protocols should guarantee secure communication in a hostile environment: they should work on “satan’s computer”.

Security protocols are different from transaction protocols, which concern the actual ***transfer*** of data, but not the ***security*** thereof.

# Example protocols

- **SSL**
  - Secure communication layer over TCP
  - Used for web transactions
- **Kerberos**
  - Authentication protocol for client/server applications with secret key cryptography
  - Keep plaintext passwords off network (as protection against “sniffers”)
- **Needham-Schroeder**
  - Famous research paper example



# Correctness of security protocols

Security protocols are notoriously hard to get right, even when they only involve a few messages per agent.

Main problems are in formalizing:

- what are the required security properties
- all that the hostile environment could possibly do

Formal methods are important for achieving precise formulations, and for exploring all scenarios



# Security Goals

- Confidentiality: only authorised people can ***see*** protected data
- Integrity: only authorised people can ***modify*** protected data; especially: no modification during transmission
- Availability: protected data is actually ***accessible*** by authorised people.
  Opposite: denial of service (DoS)

In all cases, authentication of authorised persons is crucial.

The first two are ***safety*** properties, and the last one is a ***liveness*** property.



# II. Program Correctness and Security

The area of program correctness is concerned with assuring that programs do what they should do. More precisely: that programs meet their specifications.

- In the early days of computing, program errors were thought to come from unintentional ommission, or inaccuracy.
- Nowadays, in the era of ***mobile code***, intentionally malicious code is a real (security) problem: viruses, worms, trapdoors, Trojan horses etc.

# Program Correctness versus Security

- **Correctness**
  Given ***expected*** input (precondition holds), system produces desired output
- **Security**
  Given ***any*** input, system does not:
  - reveal secrets
  - become corrupted
  - provide false guarantees

  Typically, these are ***safety*** properties.



# Program properties for security

Aspects of security can be expressed as program specifications, such as:

- (Class) invariants for expressing program safety properties, like: certain integer fields are positive, or references are non-null, to prevent exceptions.
- Modifiability clauses and reference control for integrity and confidentiality.
- Access control and information flow for integrity confidentiality. See tomorrows talk of Erik Poll.



# Modifiability

In a specification language like ***JML***, for ***Java Modeling Language*** [Leavens et al.] one may write method specifications with restrictions on the side-effects.

```
/*@ behavior
  @ requires   <precondition>
  @ modifiable <items that may be modified>
  @ ensures    <postcondition for normal termination>
  @ signals    <postcondition for exceptional terminati
  @*/
void method() { ...  }
```

Such modification restrictions, or ***frame conditions***, are important for ensuring a form of integrity.
(They also play an essential role in verification.)



# Reference control

Keeping control of pointers in programming languages is also important for confidentiality and integrity, since:

- Pointers may leak information in unintended ways.
- Even if all methods of a class maintain the class invariant, this invariant may still be disturbed by uncontrolled pointers.

Static approaches can be effective for controlling references, using suitable type annotations [Müller & Poetzsch-Heffter].

## III. Formalising Security Concepts

Three topics:

- Cryptographic primitives
- Perfect / imperfect cryptography
- Capabilities of an adversary.



## Cryptographic Primitives I

In most formalisations, cryptographic algorithms are treated as black boxes. Especially:

$$\text{Encrypt}(M, k) = \{\, M \,\}_k \qquad \text{and} \qquad \text{Decrypt}(N, k)$$

where

- $M$ is called ***plaintext***
- $\text{Encrypt}(M, k)$ and $N$ are ***ciphertext***.

(Other often-used primitive is $\text{hash}(M)$, for protecting $M$ against modification.)



## Cryptographic Primitives II

The main distinction is between:

- Symmetric or secret key systems, such as DES. A ***common*** key $k$ is used by sender and receiver, and:

$$\text{Decrypt}(\text{Encrypt}(M, k), k) = M$$

- Asymmetric or public key systems, such as RSA. Each user has two keys $k_{\text{pub}}$ and $k_{\text{priv}}$ with:

$$\text{Decrypt}(\text{Encrypt}(M, k_{\text{pub}}), k_{\text{priv}}) = M \quad \boxed{\textbf{for confidentiality}}$$

$$\text{Decrypt}(\text{Encrypt}(M, k_{\text{priv}}), k_{\text{pub}}) = M \quad \boxed{\textbf{for authentication}}$$



## Secret versus Public key

- Secret key cryptography
  - is fast (certainly when implemented in hardware), but
  - requires a special key for each pair of users.
- Public key cryptography
  - is flexible (also authentication), but
  - requires a ***reliable*** database of public keys.

Combination: use public keys for exchanging a one-time secret key for special session

# Perfectness of Cryptography

Cryptographic algorithms are not perfect: brute force attacks are possible (but should not be feasible), so there is always a chance that ciphertext may be decoded.

Should this chance be taken into account in formal approaches?

- Doing so leads to non-trivial calculi with probabilities [Mitchell].
- Ignoring these chances means that the formalisation assumes perfect cryptography. This line will be followed.



# Capabilities of an adversary I

The common model used in cryptographic protocols involves several agents that can send (encrypted) messages to each other.

One of the agents is called a Spy or Adversary, and has special powers, like observation, interception, analysis, replay etc.

Verification goal: show that the Adversary can not get a session key, cannot pretend to be someone else, or is outside the scope of a private channel, like in: $\nu K(P|Q)$.



# Capabilities of an adversary II

One of the great intrinsic problems in security analysis is: ***what capabilities does the adversary have?***

- How can you list all possible subversive actions?
- If you restrict yourself to a few of them, how realistic is your formalisation?

> Commonly used capabilities: interception, decomposition, rememberance, re-assembly, replay.



# IV. Reasoning about Security Protocols

Goal: correctness under attack.

- State exploration methods such as FDR for CSP, used by Lowe. Keeping the state space limited requires simplifications. Special case: enumerative approach of Basin, using lazy lists in Haskell.
- Belief logics formalising what agents may infer from messages received. BAN and derivatives.
- Theorem proving (especially used by Paulson), oriented towards proving safety guarantees (but absence can indicate possible attacks).

# Authentication Protocols

Authentication is very important for security: it may be used for assigning responsibility, and for giving credit.

Common distinctions:

| | ***with*** trusted party | ***without*** trusted party |
|---|---|---|
| public key | | "Needham-Schroeder" |
| secret key | Kerberos | |

The Needham-Schroeder protocol is a standard research paper example: the "alternating bit protocol" of security.

(The original paper contains several versions.)



# Needham-Schroeder protocol I

- Simple authentication protocol, introduced in 1978: two agents $A$, $B$ exchange messages in order to mutually authenticate each other, via a shared secret (which may then be used for other purposes).
- Proved "correct" in BAN-logic (1990): final beliefs are proved expressing that only $A$ and $B$ know a certain secret.
- But still incorrect, as shown by Gavin Lowe (using the FDR refinement checker for CSP) in [TACAS'96].



# Needham-Schroeder protocol II

- The protocol consists of three steps. Each step describes an event

  $$A \to B : M$$

  which states that $A$ exchanges message $M$ with $B$.
- The messages involve ***nonces***, short for ***number used once***. These nonces are essential against replay attacks, and can be seen as secrets.



# Needham-Schroeder, informally

1. $A$ wants a special session with $B$: she identifies herself, and sends a special nonce $N_A$ to $B$. This message is encrypted with $B$'s public key.
2. $B$ sends $N_A$ back to $A$, along with his own nonce $N_B$, encrypted with $A$'s public key.
   Sending $N_A$ back authenticates $B$, for only $B$ knows his private key.
3. $A$ sends $N_B$ back to $B$, encrypted with $B$'s public key.
   This also authenticates $A$.

# Needham-Schroeder, more formally

1. $A \to B : \mathsf{Encrypt}(\langle A, N_A \rangle, k^B_{\mathsf{pub}})$

2. $B \to X : \mathsf{Encrypt}(\langle N_B, N \rangle, k^X_{\mathsf{pub}})$

   upon reception by $B$ of $\mathsf{Encrypt}(\langle X, N \rangle, k^B_{\mathsf{pub}})$

3. $A \to X : \mathsf{Encrypt}(N, k^X_{\mathsf{pub}})$

   upon reception by $A$ of $\mathsf{Encrypt}(\langle N, N_A \rangle, k^A_{\mathsf{pub}})$

   after having sent $\mathsf{Encrypt}(\langle A, N_A \rangle, k^X_{\mathsf{pub}})$



# Lowe's man-in-the-middle attack

1. $A$ sends $\mathsf{Encrypt}(\langle A, N_A \rangle, k^{\mathsf{Spy}}_{\mathsf{pub}})$ to the Spy.
2. The Spy can now impersonate $A$ to an arbitrary victim $B$: it starts another run of the protocol by sending $\mathsf{Encrypt}(\langle A, N_A \rangle, k^B_{\mathsf{pub}})$ to $B$.
3. $B$ responds by sending $\mathsf{Encrypt}(\langle N_B, N_A \rangle, k^A_{\mathsf{pub}})$ to $A$.
4. $A$ receives its nonce $N_A$ that it sent to the Spy, so it replies by sending $\mathsf{Encrypt}(N_B, k^{\mathsf{Spy}}_{\mathsf{pub}})$ to the Spy.
5. The Spy now tricks $B$ by sending $\mathsf{Encrypt}(N_B, k^B_{\mathsf{pub}})$



# Result of attack

If some $A$ starts a session with the Spy, the Spy can trick any $B$ into thinking that it has a special session with $A$, whereas it has a special session with the Spy.

The protocol does not authenticate the initiator!

Fix: In the second step the responding agent's identity must be included: send $\mathsf{Encrypt}(\langle B, N_B, N \rangle, k^X_{\mathsf{pub}})$



# Needham-Schroeder formalisations

- Lowe expressed the protocol in CSP: a parallel composition of processes for agents $A$, $B$ and the Spy (non-deterministic), communicating over channels.

  Key property: a responder should commit to a session with initiator $A$ only if $A$ took part in the protocol run.

- Paulson expressed Needham-Schroeder in Isabelle, using inductive definitions of all possible
  - traces of events
  - re-assembled replay messages (sent by the Spy).

  Safety properties ("secrecy theorems") can then be proven by induction.

# Conclusions

Verification of security protocols is a real challenge, because:

- The goals are often informally stated, and are poorly understood (secrecy, authentication, anonymity, non-repudiation), so that formalisation of the verification goal is difficult
- Formalisations tend to abstract away too much, not capturing real-world attackers (doing e.g. timing or statistical analysis).
- Security is an interdisciplinary issue, and involves much more than just a protocol!



# Security work at Nijmegen

- Involve more security aspects into the current Java program verification work, especially regarding integrity and confidentiality.
- Analysis of the security implications in smart card use scenarios, e.g. in e-commerce or m-commerce, and application of specification and verification techniques in this setting.