Formal Verification of Security Protocols with ProVerif

Giann Spilere Nandi

Periodic Seminars

12 Fev 19
Outline

Introduction

ProVerif

Protocol Formal Verification - Study Case

Conclusion

Questions
Introduction

Cryptographic Protocols

Cryptographic Protocols:

- series of steps;
- message exchanges;
- hostile environment;
- security properties.
Introduction

Cryptographic Protocols

Security Properties:

- secrecy;
- authenticity;
- integrity;
- ...
Introduction
Cryptographic Protocols

Security mechanisms that are commonly used by encryption protocols:

- Public Key Encryption
- Symmetric Encryption
- Hash Functions
Introduction
Cryptographic Protocols

The effectiveness of the protocol relies on keeping in secret the keys, not the steps.
Formal methods are techniques used to model complex systems as mathematical and logical entities.
Introduction

Formal Methods Applied to Cryptographic Protocols

Aim: identify possible vulnerabilities!
Introduction

Formal Methods Applied to Cryptographic Protocols

Security Protocols:

- simple execution flow;
- difficult to design non-exploitable steps;
Introduction
Formal Methods Applied to Cryptographic Protocols

Initial research on security protocols formal verification date back from the 80’s.

Nowadays there are multiple automated formal verification tools.
ProVerif
Introduction to the Tool

ProVerif is based on the formal model (Dolev-Yao model).
Attackers are capable of:

- permeating themselves in **between the communication** of two participants in any process of the protocol;
- **modifying** and **copying** fragments of information sent in the network;
- replicating messages;
- forging messages;
Attackers are capable of:

- keeping track of **all messages** sent in the network;
- actively participating as normal agents in the protocol;
- receiving responses sent to other participants
ProVerif
Introduction to the Tool

Formal Verification of Security Protocols with ProVerif
Giann Spilere Nandi, Periodic Seminars
ProVerif

Code Organization

Public or Private
- Data Types
- Variables
- Free Names
- Constants
- Tables

Data Manipulators
- Constructors
- Destructors

Events

Local Variables
- Processes
  - Main
  - Secondary
- Queries
  - Secrecy
  - Authenticity
Protocol Formal Verification - Study Case

Saqib, 2016

Formal Verification of Security Protocols with ProVerif

Giann Spilere Nandi, Periodic Seminars

- Elliptic Curve: \( G \)
- Node's PK\(^{-1} \): \( K_x \)
- Node's PK: \( PU_x = K_xG \)

\[ \text{beginning} \]

\[ \text{before deployment} \]

\[ \text{stores the received data} \]

\[ \text{Random number: } X \]
\[ \text{Secret key: } X.G \]

\[ \{ (K_A + X)^{PU_B} \} \]

\[ \{ (K_B + Y)^{PU_A} \} \]

\[ \{ (K_B + Y)^{PU_A} \}.K_x^{-1}.PU_B = Y.G \]

\[ K_{AB} = (X.G + Y.G) \]

\[ \text{end} \]

\[ \text{after deployment} \]

\[ \text{stores the received data} \]

\[ \text{before deployment} \]

\[ \text{Random number: } Y \]
\[ \text{Secret key: } Y.G \]

\[ K_{AB} = (X.G + Y.G) \]
Protocol Formal Verification - Study Case
Saqib, 2016 - Model Breakdown

Data Types

1. type skey.
2. type secretkey.
3. type id.
4. type pkey.
Protocol Formal Verification - Study Case
Saqib, 2016 - Model Breakdown

1. free C1: channel.
2. free C2: channel.
3. free CX: channel.
4. const GP: g [private].
5. table alreadyPaired(id, id).
Protocol Formal Verification - Study Case
Saqib, 2016 - Model Breakdown

1. fun encryption(g,pkey): g.
2. reduc forall KA': skey, X': number, GENPOINT: g, KB': skey, PUA': pkey;
   ↪ decryption(encryption(addsSkeyPlusNumber(KA',X'), calcPublicKey
   ↪ (KB', GENPOINT)), KB', calcPublicKey(KA', GENPOINT)) =
   ↪ calcSecretKey(X', GENPOINT).
3. fun calcSharedKey(secretkey, secretkey): secretkey.
let nodeA() =
  new KA: skey; new IDA: id;
  let PUA = calcPublicKey(KA, GP) in
  insert publicKeysA(IDA, PUA); out(CX, PUA);
  !(get publicKeysB(IDBR1: id, PURB: pkey) in
    get alreadyPaired(=IDA, =IDBR1) in (event eNodesAlreadyPairedA(
      IDA, IDBR1))
    else (new X: number;
      let XG = calcSecretKey(X, GP) in
      event eNodeACreatesTheSecretKey(XG);
      let KAXPUB = encryption( addsSkeyPlusNumber(KA, X), PURB) in
      event eSendKAXPUB(KAXPUB);
      out(C1, (KAXPUB, IDA, IDBR1));
      in(C2, (K2:g, IDBR2: id, IDAR: id));
      if ((IDAR = IDA) && (IDBR2 = IDBR1)) then
        let GY = decryption(K2, KA, PURB) in(
          event eDecryptedUsingKA(GY,KA);
          let XGGY = calcSharedKey(XG, GY) in
          event eNodeAComputesSharedKey(XGGY)))
  )
).
Protocol Formal Verification - Study Case
Saqib, 2016 - Model Breakdown

1. process
2. (!nodeB() | !nodeA())
Unauthorized agents are not capable of deriving specific information...
Protocol Formal Verification - Study Case
Saqib, 2016 - Secrecy Queries

Queries

1. query secret KAXPUB.
2. query secret KBYPUA.
3. query attacker(new KA).
4. query attacker(new KB).

Secrecy

Authenticity
Correspondence Assertions

"if an event e has been executed, then e’ has been previously executed"
<table>
<thead>
<tr>
<th>Aliveness</th>
<th>Weak Agreement</th>
<th>Non-injective Agreement</th>
<th>Injective Agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Protocol Formal Verification - Study Case
Saqib, 2016 - Authenticity Queries

Queries

Secrecy

Authenticity

1. query \( q_{XGGY}: \text{secretkey}, q_{YGGX}: \text{secretkey}, q_{KBYPUA}: g, q_{KAXPUB}: g, q_{KB}: \leftarrow \text{skey}, q_{KA}: \text{skey}, q_{Y}: \text{number}, q_{X}: \text{number}, q_{GX}: \text{secretkey}, q_{GY}: \leftarrow \text{secretkey}; \) event(eNodeAComputesSharedKey(q_{XGGY})) \( \implies \) (inj-event (event(eNodeBComputesSharedKey(q_{YGGX})) \( \implies \) inj-event (eSendKAXPUB(q_{KAXPUB}))) \&\& \) q_{XGGY} = calcSharedKey(q_{GX}, q_{GY}) \&\& \( q_{YGGX} = \text{calcSharedKey}(q_{GX}, q_{GY}) \&\& q_{GX} = \text{calcSecretKey}(q_{X}, GP) \&\& q_{GY} = \text{calcSecretKey}(q_{Y}, GP) \&\& q_{KBYPUA} = \text{encryption}(\text{addsSkeyPlusNumber}(q_{KB}, q_{Y}), \text{calcPublicKey}(q_{KA}, GP)) \&\& q_{KAXPUB} = \text{encryption}(\text{addsSkeyPlusNumber}(q_{KA}, q_{X}), \text{calcPublicKey}(q_{KB}, GP)).}
Did the symmetric key calculated by A contain the secret key calculated by B, and did the symmetric key calculated by B actually contain the secret key sent by A?
Protocol Formal Verification - Study Case
Saqib, 2016

<table>
<thead>
<tr>
<th>BASE SERVER</th>
<th>NODE A</th>
<th>NODE B</th>
</tr>
</thead>
</table>
| **Elliptic Curve:** G  
**Node's PK:** Kx  
**Node's PK:** PUx = KxG | stores the received data | stores the received data |
| **Random number:** X  
**Secret key:** X.G | (**Kx+X**)PUB | (**Kx+X**)PUB |
| (**Kx+X**)PUB | (**Kx+X**)(PUA-1)PUB = X.G | (**Kx+X**)(PUA-1)PUB = X.G |
| (**Kx+X**)(PUA-1)PUB = X.G | **Random number:** Y  
**Secret key:** Y.G | (**KB+Y**)PUA |
| (**KB+Y**)PUA | (**KB+Y**)(PUA-1)PUA = Y.G | (**KB+Y**)(PUA-1)PUA = Y.G |
| (**KB+Y**)(PUA-1)PUA = Y.G | **KAB = (X.G + Y.G)** | (**KB+Y**)(PUA-1)PUA = Y.G |
| **KAB = (X.G + Y.G)** | | **KAB = (X.G + Y.G)** |

**Formal Verification of Security Protocols with ProVerif**
Giann Spilere Nandi, Periodic Seminars
Protocol Formal Verification - Study Case
Saqib, 2016 - Result Analysis

Formal Verification of Security Protocols with ProVerif
Giann Spilere Nandi, Periodic Seminars
Conclusion

”Take Home” Messages

- It is **difficult** to design security protocols with **no vulnerabilities**;
- Formal verification techniques **help** on the process of checking if protocols guarantee certain **security properties**;
- ProVerif is one of the available tools to automate formal verification;
Conclusion

"Take Home" Messages

- ProVerif can only verify what the user provides to it.
- Find the perfect balance between the levels of detail and abstraction for your model;
- The results given by ProVerif should serve as a tool to improve the analysis of the protocol.
Questions?