Modelling and Analysis of a Hierarchy of Distance Bounding Attacks

Tom Chothia, Joeri de Ruiter and Ben Smyth
Introduction

• A unified framework for distance bounding attacks.

• Examples: Contactless EMV & NXP’s DB protocol.

• A modelling language for DB protocols.

• A hierarchy of security properties, matched to particular attacker models.

• Automatically checking previously defined symbolic properties.
Core EMV Protocol

Shop

UN, amount, currency, ...

Card

Kbank: 3-DES key shared with bank
Kcard: an RSA public
Kcert: Bank cert. for Kcard

Generate nonce: Nc
Session key based on ATC: $K_s = \text{Enc}_{K_{\text{bank}}}(\text{ATC})$

MAC$_{K_s}(\text{amount}, \text{currency}, \text{UN},...)$

Sign$_{K_{\text{card}}}(\text{amount}, \text{currency}, \text{UN}, \text{Nc},...)$

Kcert, ATC

AC, ATC

Bank’s Verification key
Master-card’s PayPass

Diagram showing the process of selecting a PayPass AID, reading the record, generating an AC (Access Condition) with a random number and transaction details, encrypting ATC with a key derived from PubCA, and signing the SDAD (Secure Document Access Data) with the private PayPass key.
Shop

Phone1

Phone2

Card

SELECT

AIDs

SELECT AID

AIDs

GPO

READ1

Static data

READ2

ATC, AC, SDAD, PAN

AC, SDAD

UN, amount

Nc

SSAD, Nc

SELECT

AIDs

GPO

ATC, AC, SDAD, PAN

READ2

SSAD, Nc

Only added time delay

↓↑
MasterCard’s Relay Resistance Protocol (RRP) (similar to PaySafe)

- Uses New Command
- Timing profile sent by card
- We check this as auth. property
NXP distance bounding protocol

• NXP sell a distance bounding smart card.

• NXP have patented a distance bounding 😊

• Patent documents are really hard to read 😞

“This need may be met by the subject matter according to the independent claims. Advantageous embodiments of the present invention are set forth in the dependent claims.”
NXP Protocol.

Only in one version

Can be split into 8 one bytes message
Some Questions

• How can we formally (symbolically) define these protocols?

• How can we say if these protocols are “secure”?

• What does “secure” even mean in this context?
Our modelling language for DB

\[
in (x).P \\
out <x>.P \\
P \mid Q \\
!P \\
new a.P \\
let x = D in P else Q \\
event(X).P \\
startTimer.P \\
stopTimer.P \\
\]

Locations: \( L = [ P ] \) or \( L \mid L \)

Eg.
\[
[ \text{EMVCard} ] \mid [ \text{ShopReader} ] \\
[ \text{EMVCard} \mid \text{ShopReader} ]
\]
let Verifier =
  out c<SELECT,AID>.
  in c(pdol).
  new UN.
  out c<GET_PROCESSING_OPTIONS,UN,amount>.
  in c(aip,afl,NC).
  out c<GENERATE_AC>.
  in c(SDAD,AC).
  out c<READ_RECORD>.
  in c(cCert).
  let cKey, cId = checksign(cCert,getPubKey(BANK_ID)) in
  let (=UN,=NC,=rAmount,ATC,AC)=checksign(SDAD,cKey) in
  event Verified(cId).
let Verifier =
  out c<SELECT,AID>.
in c(pdol).
new UN.
out c<GET_PROCESSING_OPTIONS,UN,amount>.
in c(aip,afl,NC).
startTimer. out c<GENERATE_AC>.
in c(SDAD,AC). stopTimer.
out c<READ_RECORD>.
in c(cCert).
let cKey, cId = checksign(cCert, getPubKey(BANK_ID)) in
let (=UN,=NC,=rAmount,ATC,AC)=checksign(SDAD,cKey) in
event Verified(cId).

Verifiers = !(new amount.!Verifier)
Provers = !(new id. let idP = id in
  let cCert = sign(getPubKey(idP), idP),
      getPrivKey(BANK_ID)) in
  !event Start(idP). Prover ]

Unbounded number ids each for an unbounded number of runs
StartTimer blocks an messages from remote locations

startTimer
challenge
response
stopTimer

stopTimer re-enables messages from remote locations

startTimer
challenge
response
stopTimer
Key observation: The semantics just needs to block outputs from remote locations while a timer is running

We write \[
\left[ \text{Process} \right]^{\text{number of timers running}}
\]

\[
\left[ \text{in c(x).P | out c<n>.Q} \right]_r \rightarrow \left[ \text{P}\{n/x\} | Q \right]_r
\]

\[
\left[ \text{out c<n>.Q} \right]_r | [P]_0 \rightarrow [Q]_r | [\text{out c<n>!P}]_0
\]

\[
\left[ \text{out c<n>.Q} \right]_r \rightarrow \left[ \text{out c<n> | Q} \right]_r
\]
Definitions for the symbolic literature
Relay/Mafia Fraud: attackers relay and interfere with messages
Distance Fraud: remote dishonest prover tricks the verifier
Distance Hijacking: remote dishonest prover uses a local honest prover
Terrorist Fraud: A remote dishonest prover* and local attacker
Assisted Distance Fraud: remote dishonest prover* and local dishonest prover
Definitions for the symbolic literature

• Relay/Mafia Fraud: attackers relay and interfere with messages

• Lone Distance Fraud: remote dishonest prover tricks the verifier

• Distance Hijacking: remote dishonest prover uses a local honest prover

• Terrorist Fraud: A remote dishonest prover* and local attacker

• Assisted Distance Fraud: remote dishonest prover* and local dishonest prover
Relay Attack

- There exists relay attack against the protocol P and V if there exists A such that

\[ [V(id)|A] | [P(id)|A] \]

I.e.

\[
\begin{align*}
  [V | A] & | [P(id) | A] \\
  \rightarrow^* [X] & | [\text{new id}.Q | Y] \\
  \rightarrow [X] & | [Q{a/id} | Y] \\
  \quad & | \text{[event verified(a).R | W]} | [Z]
\end{align*}
\]
Distance Fraud

- Dishonest prover DP-A(id) = ![new id.<board cast all secret values>] | A

- **Lone Distance Fraud**: A dishonest prover remotely authenticates to a verifier.

  \[ V(id) | [\text{DP-A(id)}] \]

- **Distance Hijacking**: remote dishonest prover uses a local honest prover

  \[ [V(id)|P(id')] | [\text{DP-A(id)}] \]

E.g.: For RRP:

\[
\begin{align*}
\text{DP-A(id)} &= A | ![\text{new id. out c<id>!}] \\
&= \text{let cert} = \text{sign}((\text{getPubKey(id), id}), \text{getPrivKey(BANK_ID)}) \text{ in} \\
&= \text{out c<getPrivKey(id), cert, sharedKey(id)>).}
\end{align*}
\]
Terrorist Frauds

E.g.: For RRP:

TP-A(id) = A | ! new id. out c<id>. ( ! in c (atc, message);
  let macKey=genKey(atc, sharedKey(idP)) in
  let messageMAC = mac(message, macKey) in
  out c<messageMAC>
    | ! in c(message);
    let signed=sign(message, getPrivKey(id)) in
    out c<signed>
    | out c<cardCert, id>.

[V(id)|DP-A(id')] [[TP-A(id)]]
Our Building Blocks

- Arbitrary number of provers
- Verifier looking for one of “id”
- A Dolev–Yao attacker
- Other Provers

Trying to trick verifier

- A dishonest prover
- A terrorist prover

Verifier doesn’t care about

- A dishonest prover
- A terrorist prover
Ordering the Properties

• Our building blocks form a hierarchy.

• Each level is strictly more expressive than the one below.

• Replacing any process with the one above it, at a particular location, makes the attacker more powerful.
Equalities between processes

\[ [V(id)|A] \mid [P(id)|A] \]

\[ = \]

\[ [V(id)|A] \mid [P(id)|A|P(id')] \]
Some Heuristics

\[ [V(id)] \mid [TP-A(id)] \]

\[ \equiv [V(id)] \mid [DP-A(id)] \]
Distance Fraud

Mafia fraud/Relay

Uncompromised Distance Bounding

Relay Hijacking

Assisted Distance Fraud

Terrorist Fraud

Distance Hijacking

Distance Fraud

Key:

P(id): honest provers with identity “id”
V(id): verifier wishing to verifier “id”
A: attacker process
TP-A(id): terrorist provers, acting as “id”
DP-A(id): dishonest provers, acting as “id”
Uncompromised Distance Bounding

Relay Hijacking
Distance Fraud
[V(id)|DP-A(id')] | [TP-A(id)]

Mafia fraud/Relay
[V(id)|A] | [P(id)|A]

Terrorist Fraud
[V(id)|A] | [TP-A(id)]
[TP-A(id)] | [P(id)|A]

Assisted Distance Fraud
[V(id)|P(id')|A] | [TP-A(id)]

Distance Hijacking
[V(id)|A] | [P(id)|TP-A(id')]

Uncompromised Distance Bounding
[V(id)|P(id')|A] | [P(id)|A]
[V(id)|A] | [P(id)|P(id')]
[V(id)|P(id')|A] | [P(id)|P(id')]

Remote and local attacker
Remote attacker only
Some untrusted devices
Trusted devices only

No terrorist attacker
Terrorist attacker

Key:
P(id): honest provers with identity “id”
V(id): verifier wishing to verifier “id”
A: attacker process
TP-A(id): terrorist provers, acting as “id”
DP-A(id): dishonest provers, acting as “id”

Terrorist attacker
No terrorist attacker
Remote and local attacker
Remote attacker only
Some untrusted devices
Trusted devices only

Trusted devices only
Some untrusted devices
Remote and local attacker
Remote attacker only
Uncompromised Distance Bounding
[V(id)|DP-A(id')] | [P(id)|DP-A(id')]

Key:
P(id): honest provers with identity “id”
V(id): verifier wishing to verifier “id”
A: attacker process
TP-A(id): terrorist provers, acting as “id”
DP-A(id): dishonest provers, acting as “id”

Prover being checked is compromised
Prover being checked is not compromised
No terrorist attacker
Terrorist attacker
Remote and local attacker
Remote attacker only
Some untrusted devices
Trusted devices only

Mafia fraud/Relay
[V(id)|A] | [P(id)|A]

Relay Hijacking
[V(id)|P(id')|A] | [P(id)|A]

Assisted Distance Fraud
[V(id)|DP-A(id')] | [TP-A(id)]

Terrorist Fraud
[V(id)|A] | [TP-A(id)]

Distance Hijacking
[V(id)|A] | [TP-A(id)]

Distance Fraud
[V(id)] | [P(id)|DP-A(id')]

Terrorist attacker
No terrorist attacker
Trusted devices only
Remote and local attacker
Remote attacker only
Some untrusted devices
Automatically Checking

• We translate our DB calculus into the applied pi-calculus, and use ProVerif to check processes automatically.

• The translation uses 3 phases:
  • Phase 1, before the timer start
  • Phase 2, while the timer is running
  • Phase 3, after the time stops.

startTimer jumps from phase 1 to phase 2.
stopTimer jumps from phase 2 to phase 3.

Process at the same location as the verifier can act in all phases
Process at a different location can only act in Phase 1 and Phase 2.
Demo
<table>
<thead>
<tr>
<th></th>
<th>Mafia Fraud / Relay</th>
<th>Uncompromised Distance Bounding</th>
<th>Distance Fraud</th>
<th>Terrorist Fraud</th>
<th>Timing information authenticity</th>
</tr>
</thead>
<tbody>
<tr>
<td>PaySafe</td>
<td>OK</td>
<td>OK</td>
<td>Attack</td>
<td>Attack</td>
<td>N/A</td>
</tr>
<tr>
<td>PaySafe with changes [28]</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>Attack</td>
<td>N/A</td>
</tr>
<tr>
<td>MasterCard’s RRP</td>
<td>OK</td>
<td>OK</td>
<td>Attack</td>
<td>Attack</td>
<td>OK</td>
</tr>
<tr>
<td>NXP’s protocol (unique keys)</td>
<td>OK</td>
<td>OK</td>
<td>Attack</td>
<td>Attack</td>
<td>OK</td>
</tr>
<tr>
<td>NXP’s protocol (global key)</td>
<td>OK</td>
<td>Attack</td>
<td>Attack</td>
<td>Attack</td>
<td>OK</td>
</tr>
<tr>
<td>NXP’s variant 1 (unique keys)</td>
<td>OK</td>
<td>OK</td>
<td>Attack</td>
<td>Attack</td>
<td>N/A</td>
</tr>
<tr>
<td>NXP’s variant 2 (unique keys)</td>
<td>OK</td>
<td>OK</td>
<td>Attack</td>
<td>Attack</td>
<td>N/A</td>
</tr>
<tr>
<td>Meadows et al. [30]</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>Attack</td>
<td>N/A</td>
</tr>
<tr>
<td>MAD (One-Way) [36]</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>Attack</td>
<td>N/A</td>
</tr>
<tr>
<td>CRCS [32]</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>Attack</td>
<td>N/A</td>
</tr>
<tr>
<td>Hancke and Kuhn [24]</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>N/A</td>
</tr>
<tr>
<td>Poulidor [35]</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>N/A</td>
</tr>
<tr>
<td>Tree-based [5]</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>N/A</td>
</tr>
<tr>
<td>Uniform [29]</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Conclusion

• A unified framework for distance bounding attacks.

• Examples: Contactless EMV & NXP’s DB protocol.

• A modelling language for DB protocols.

• A hierarchy of security properties, matched to particular attacker models.

• Automatically checking previously defined symbolic properties.