Forbidden City Model –
towards a Practice Relevant Framework for Designing Cryptographic Protocols

Mirosław Kutyłowski
joint work with Lucjan Hanzlik, Kamil Kluczniak, Przemysław Kubiak, Łukasz Krzywiecki

Wrocław University of Technology, Poland

ISPEC 2014, Fuzhou,
Huawei Information Security Practice and Experience Forum
Introduction
Constructing a security solution

### Pipelining approach

1. **design a cryptographic scheme**
   - design an algorithm
   - formulate a security model
   - prove security

2. **implement**
   - obey the rules for secure product development
   - certification of the product

3. **deploy**
   - get a high quality certified product
   - install
   - run system audits
Constructing a security solution

What goes wrong?

1. design a cryptographic scheme
   - model driven by properties of the algorithm
   - focusing on less relevant issues
   - ignored threats

2. implement
   - gap between cryptographic schemes and implementation environment – unmatched assumptions
   - protection profiles incomplete, so certification useless
   - inspecting documentation and not the reality

3. deploy
   - certificates and standards regarded as security proofs
   - audits: ≈ buying a legal insurance, not a real inspection
(In)security in practice

user: somebody forged my signature and took away my money!!
(In)security in practice

user: somebody forged my signature and took away my money!!

cryptographer: sorry but the attacker performed an attack that is outside the scope of the security model
user: somebody forged my signature and took away my money!!

cryptographer: sorry but the attacker performed an attack that is outside the scope of the security model

engineer: I have implemented exactly as it was specified. I must not make any adjustments or extensions.
(In)security in practice

user: somebody forged my signature and took away my money!!

cryptographer: sorry but the attacker performed an attack that is outside the scope of the security model

engineer: I have implemented exactly as it was specified. I must not make any adjustments or extensions.

certification authority: the certificate concerns the protection profile. The properties stated there are fulfilled. The problem is outside the scope covered by the PP.
user: somebody forged my signature and took away my money!!

cryptographer: sorry but the attacker performed an attack that is outside the scope of the security model

engineer: I have implemented exactly as it was specified. I must not make any adjustments or extensions.

certification authority: the certificate concerns the protection profile. The properties stated there are fulfilled. The problem is outside the scope covered by the PP.

system owner: all stated conditions concerning running the system have been obeyed. Certificates have been checked, audits performed. We could not do more.
(In)security in practice

user: somebody forged my signature and took away my money!!

cryptographer: sorry but the attacker performed an attack that is outside the scope of the security model

engineer: I have implemented exactly as it was specified. I must not make any adjustments or extensions.

certification authority: the certificate concerns the protection profile. The properties stated there are fulfilled. The problem is outside the scope covered by the PP.

system owner: all stated conditions concerning running the system have been obeyed. Certificates have been checked, audits performed. We could not do more.

auditor: Everything has been checked. The system is in the state claimed. The audit is not responsible for checking if security if thereby guaranteed.
End-to-end systems

E2E design

- all components of the system described
- clear assumptions about threats for each component
- security analysis taking all threats into account

Lessons learnt from design for E2E systems

- the components have to be designed in accordance to the global goals
- securing against all threats at once is much different than securing them one by one
- new approaches
The ratio
\[
\frac{\text{number of algorithms}}{\text{number of models}}
\]
is a small constant. The models are driven by necessity to publish and not to solve a practical problem.

Relations between the models becomes a hard research topic itself.

Some wonderful concepts are too advanced to be understood and used by average system designers. example: universal composability
Models and Standards

Academic security track

- provide a model to be able to “sell” the algorithm for a publication
- model has to be advanced as well as the proof ...
- ... however not too long due to the page limit (maybe 50 page appendix)

Industrial track

- obtain a patent
  – obscure the details in order to make future claims more easy
- go through standardization process
- provide some formal model and security considerations in order to get a high EAL level
there is no mathematical argument to claim that security proofs can be short and interesting

- academics prefer to read short and clever paper
- hard to get an academic degree for a boring and routine dissertation, no journal publication

industry pays for this tedious work only if the investment necessary for selling products
Embedded security devices
Cryptography in use

Large scale use of cryptography:

- telecommunication (SIM cards, AuC, encryption, . . .)
- ATM
- biometric passport, eID documents
- HSM based services
- storage encryption
- . . .

mostly the classical models not focused on peculiarities of such embedded systems
Challenges for embedded systems

Black box devices

- the device must be sealed in some sense in order to prevent reading the internal state (ephemeral keys, long time secret keys, . . .)
- the operations should be atomic in the sense that only the final outcome should be available

however: plenty ways to cheat the user of a black-box device (malicious cryptography):
- subliminal channels,
- kleptography,
- PRNG with seeds known for outsiders,
  . . .
Code replacement

Smart cards
- critical parts that are fixed for the whole population of devices in ROM
- ROM cannot be overwritten

Consequences
- no virus can overwrite ROM memory, but
- if an error occurs in the protocol, there might be attempts to keep it secret, to deny the consequences, …
- you must be extremely careful when designing the solution: it must survive years (e.g. 10 years for ID cards)
Restarting attacks

Smart card

- energy from the reader, the reader may switch off the card at any time
- the card may not remember that there have been attempts with broken sessions
- solutions like: first increase some counter in EEPROM and then execute the operation
Leakage

Sources of leakage

- traditional approach - limited amount of bits
- smart card: EEPROM, operating system, side channel
- the user himself can try to break into own device (e.g. in order to deny own previous operations)
Time complexity

Cryptographic coprocessors

- some fixed set of (advanced) operations implemented on a cryptographic processor
- otherwise very poor computational resources (e.g. a slow 8bit processor)
- the choice of operations is a matter of historic development

Consequences

- practical computational complexity might be very different from the mathematical computational complexity,
  multiplication of long numbers performed via RSA
- complexity analysis should take into account industrial standards
Space complexity

Smart cards’ memory

- space available is very limited
- limitations not only on the size of permanent storage for keys but also for the storage of intermediate results and for the program code
- reusing the same code blocks for different purposes can be very useful

consequences

- discrepancy between the theory and practice for cryptographic protocols
- any paper that compares the size of executable code?
- security depends on how many features have to be eliminated due to the lack of space
Randomness and reality

RNG
- aging problems
- hardware Trojans
- unknown physical effects
- quality-feasibility tradeoff
- replacing by another source in a black box device
- acceptance based alone on passing NIST tests

PRNG
- provable properties
- based on secrecy of the seed (but for digital signatures we anyway depend on secure storage)

what really counts is computationally random and not information-theoretically random
Models
Example - authenticated key exchange

Goal

- the communicating parties perform mutual authentication
- at the end the parties share a session key
- nobody else learns the key, the session key known only to the communicating parties
- recent requirements: privacy
  - passive eavesdroppers
  - active adversaries
  - selling data to other parties

most protocols in the literature first exchange ID data in clear! (but ICAO standards are privacy aware)
Citation from *Examining Indistinguishability-Based Proof Models for Key Establishment Protocols* by Choo & Boyd & Hitchcock

Adversarial Powers. . . . the adversary $A$ is defined to be a probabilistic machine that is in control of all communications between parties via the predefined oracle queries described below:

**Send:** This query computes a response according to the protocol specification and decision on whether to accept or reject yet, and returns them to $A$.

**Session-Key Reveal**($U_1, U_2, i$): Oracle $\Pi_{U_1, U_2, i}$, upon receiving a Session-Key Reveal query, and if it has accepted and holds some session key, will send this session key back to $A$. This query is known as a Reveal($U_1, U_2, i$) query in the Bellare-Rogaway models.

**Session-State Reveal:** Oracle $\Pi_{U_1, U_2, i}$, upon receiving a Session-State Reveal($U_1, U_2, i$) query and if it has neither accepted nor held some session key, will return all its internal state (including any ephemeral parameters but not long-term secret parameters) to $A$.

**Corrupt:** Corrupt($U_1, KE$) query allows $A$ to corrupt the principal $U_1$ at will, and thereby learn the complete internal state of the corrupted principal. The corrupt query also gives $A$ the ability to overwrite the long-lived key of the corrupted principal with any value of her choice (i.e. $KE$).

**Test:** Test($U_1, U_2, i$) query is the only oracle query that does not correspond to any of $A$’s abilities. If $\Pi_{U_1, U_2, i}$ has accepted with some session key and is being asked a Test($U_1, U_2, i$) query, then depending on a randomly chosen bit $b$, $A$ is given either the actual session key or a session key drawn randomly from the session key distribution.

further notions:

- partnership
- session freshness
Problems

Unclear picture

- there are many models extending CK to cover more and more sophisticated attack scenarios
- the model is focused on two-party communication with the man-in-the-middle who may sometimes break in
- how to explain the hardware engineer e.g. what is a “fresh session”?
  (intention: call to the oracles that obviously reveal the sensitive information)
Model for co-design of crypto protocols and crypto systems

Component
- storage inside
- code inside
- interface
- attack possibilities

Nesting
- a component may contain internal components
Forbidden City Model

Kutylowski et al.

introduction

reality

models

embedded security

attack opportunities

complexity

randomness

models

traditional

Forbidden City

examples

Schnorr signature

authentication

discussion

Example
Password authentication

Forbidden City Model
Kutyłowski et al.

introduction
reality
models
embedded
security
attack opportunities
complexity
randomness
models
traditional
Forbidden City
examples
Schnorr signature
authentication
discussion
Examples
Schnorr signature

1. choose $k \in [1, q - 1]$ uniformly at random,
2. $r := g^k$,
3. $e := H(M || r)$,
4. $s := k - x \cdot e \mod q$,
5. return $(r, s)$.
It does not make sense to take RNG outside the signing unit – as $k$ leaks $x$ immediately
Schnorr signature

placement of HASH

- if \( M \) to long to be uploaded then better change the standard:
  \[
e := H(H(M) || r)
\]

- computing \( e := H(r || M) \) looks better from cryptographic point of view but is risky at implementation: long delay after computing \( r \)
Distributed Schnorr signature

1. Choose \( k_1 \in [1, q - 1] \) uniformly at random.
2. \( r_1 := g^{k_1} \)
3. \( \text{count} := \text{count} + 1 \)
4. \( k_2 := \text{PRNG}(z, \text{count}) \mod q \)
5. \( r_2 := g^{k_2} \)
6. \( r := r_1 \cdot r_2 \)
7. \( e := \text{Hash}(M \| r) \)
8. \( s_1 := k_1 - x_1 \cdot e \mod q \)
9. \( s_2 := k_1 - x_2 \cdot e \mod q \)
10. \( s := s_1 + s_2 \mod q \)
11. Return \((e, s)\)
Distributed Schnorr signature

- Forbidden City Model
- Kutyłowski et al.
- Introduction
  - Reality models
  - Embedded security
  - Attack opportunities
  - Complexity
  - Randomness
- Models
  - Traditional
  - Forbidden City
- Examples
  - Schnorr signature
  - Authentication
- Discussion

Diagram:
- Sign-RNG
- Sign-PRNG
- RNG
- PRNG
- HASH
- MAIN
- PC
- Signing device for Schnorr
Distributed Schnorr signature

code executed by SIGN-RNG:
activate upon request from MAIN
choose \( k_1 \in [1, q) \) uniformly at random
\( r_1 := g^{k_1} \)
send \( r_1 \) to MAIN
receive \( e \) from MAIN
\( s_1 := k_1 - x_1 \cdot e \mod q \)
send \( s_1 \) to MAIN
reset

code executed by SIGN-PRNG:
activate upon request from MAIN
send a request to PRNG
receive \( k_2 \) from PRNG
\( r_2 := g^{k_2} \)
send \( r_2 \) to MAIN
receive \( e \) from MAIN
\( s_2 := k_1 - x_2 \cdot e \mod q \)
send \( s_1 \) to MAIN
reset

code executed by MAIN:
send a request to SIGN-RNG
send a request to SIGN-PRNG
receive \( r_1 \) from SIGN-RNG
receive \( r_2 \) from SIGN-PRNG
\( r := r_1 \cdot r_2 \)
receive pre-hash \( h \) from the external HASH
send \( h, r \) to the internal HASH
receive \( e \) from the internal HASH
send \( e \) to SIGN-RNG
send \( e \) to SIGN-PRNG
receive \( s_1 \) from SIGN-RNG
receive \( s_2 \) from SIGN-RNG
\( s := s_1 + s_2 \)
send \((s, e)\) as the output

code executed by PRNG:
receive request from SIGN-PRNG
\( count := count + 1 \)
\( k_2 := PRNG(z, count) \mod q \)
send \( k_2 \) to SIGN-PRNG
Target

- the reader shows the chip that it knows the password
- the chip authenticates itself with an asymmetric algorithm
- the minimal changes to PACE - backwards compatibility
### Version 1

<table>
<thead>
<tr>
<th>Card</th>
<th>Reader</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi, x_A, X_A = g^{x_A} )</td>
<td>( \pi )</td>
</tr>
<tr>
<td>random ( s ) chosen</td>
<td>ENC(( K_{\pi}, s ))</td>
</tr>
<tr>
<td>choose ( y_A \leftarrow \mathbb{Z}_q^* )</td>
<td>retrieve ( s )</td>
</tr>
<tr>
<td>( Y_A := g^{y_A} )</td>
<td>choose ( y_B \leftarrow \mathbb{Z}_q^* )</td>
</tr>
<tr>
<td>abort if ...</td>
<td>( Y_A \rightarrow )</td>
</tr>
<tr>
<td>( h := Y_A^{y_B} )</td>
<td>abort if ...</td>
</tr>
<tr>
<td>( \hat{g} := h \cdot g^s )</td>
<td>( h := Y_A^{y_B} )</td>
</tr>
<tr>
<td>choose ( y'_A \leftarrow \mathbb{Z}_q^* )</td>
<td>( \hat{g} := h \cdot g^s )</td>
</tr>
<tr>
<td>( Y'_A := \hat{g}^{y'_A} )</td>
<td>choose ( y'_B \leftarrow \mathbb{Z}_q^* )</td>
</tr>
<tr>
<td>check ...</td>
<td>( Y'_A \rightarrow )</td>
</tr>
<tr>
<td>( K_{\ldots} := H(\ldots</td>
<td></td>
</tr>
<tr>
<td>...tags checked</td>
<td>( Y_A \rightarrow )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>check ...</td>
</tr>
<tr>
<td>( K_{\ldots} := H(\ldots</td>
<td></td>
</tr>
</tbody>
</table>

\[ w := y_A / x_A \]

\[ E_{K_{_{SC}}}'(w, \text{cert}_A) \rightarrow \text{decrypt with } K_{_{SC}}' \]

check certificate \( \text{cert}_A \)

abort if \( X_A^w \neq Y_A \)
<table>
<thead>
<tr>
<th>Card</th>
<th>Reader</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi, x_A, X_A = g^{x_A}$</td>
<td>$\pi$</td>
</tr>
<tr>
<td>random $s$ chosen</td>
<td>$\text{ENC}(K_\pi, s)$</td>
</tr>
<tr>
<td>choose $y_A \leftarrow \mathbb{Z}_q^*$</td>
<td>retrieve $s$</td>
</tr>
<tr>
<td>$Y_A := X_A^{y_A}$</td>
<td></td>
</tr>
<tr>
<td>abort if ...</td>
<td>$Y_A \rightarrow$</td>
</tr>
<tr>
<td>$\hat{h} := Y_B^{y_A}, h := \hat{h}^{x_A}$,</td>
<td>abort if ...</td>
</tr>
<tr>
<td>$\hat{g} := h \cdot g^s$</td>
<td>$h := Y_A^{y_B}$</td>
</tr>
<tr>
<td>choose $y'_A \leftarrow \mathbb{Z}_q^*$</td>
<td></td>
</tr>
<tr>
<td>$Y'_A := \hat{g}^{y'_A}$</td>
<td>$Y'_B := \hat{g}^{y'_B}$</td>
</tr>
<tr>
<td>check ...</td>
<td>$Y'_A \rightarrow$</td>
</tr>
<tr>
<td>$K_{\ldots} := H(\ldots</td>
<td></td>
</tr>
<tr>
<td>...tags checked</td>
<td>$K_{\ldots} := H(\ldots</td>
</tr>
<tr>
<td>$w := y_A$</td>
<td>$E_{K'_SC}(w, \text{cert}_A)$</td>
</tr>
<tr>
<td>$w := y_A$</td>
<td>decrypt with $K'_SC$</td>
</tr>
<tr>
<td>check certificate $\text{cert}_A$</td>
<td></td>
</tr>
<tr>
<td>abort if $X_A^w \neq Y_A$</td>
<td></td>
</tr>
</tbody>
</table>
## Differences between Versions 1 and 2

### Standard compilation
- exactly equivalent computations
- if the executable code generated automatically by a compiler, it is likely that they end up with the same code
- however: version 1 leaks $x_A$ in case of ephemeral key leakage

### Architecture
- **Version 1:** it does not make sense to protect $x_A$ in an internal component, as it can be derived from $y_A$ and $w$
- **Version 2:** $x_A$ in the component that only performs the mapping $z \rightarrow z^{x_A}$

### Ephemeral key leakage
- **Version 1:** $x_A$ leaked
- **Version 2:** $x_A$ remains secure
Discussion
Divide and conquer

How we make the situation easier?

Example: $z \rightarrow z^{x_A}$ component

- the component can be considered separately
- whatever happens outside, this is only a component that is an oracle for $z \rightarrow z^{x_A}$

unfortunately impossible for DSA but immediate for the BLS signatures
Number of attack scenarios

A protocol with $k$ messages

- the number of scenarios which messages to manipulate - already $2^k$ possibilities
- the number of possibilities how to manipulate a single step: $\ldots$

Very quickly it becomes infeasible to perform analysis due to the number of cases, unless some reduction of cases found.
Adversary types

Adversary type

- specification of
  - possible attack methods
  - known data

Attack

for a given adversary type perform an attack and gain:
- new possibilities to attack
- new data learnt via attack (e.g. crossing the component boundaries via an cryptanalytic attack)

Graph

attacks defines edges between the adversary types
we look for the transitive closure in this graph
One adversary is not enough

the situation is even more complicated:

<table>
<thead>
<tr>
<th>Attack scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>- actors: a smart card $C$, a terminal $T$, Eve</td>
</tr>
<tr>
<td>- goal: the terminal $T$ wants to convince Eve that it has interacted with $C$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-monotonicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>- It is not true anymore that if $T$ has more resources then it can easier attack:</td>
</tr>
<tr>
<td>- with certain secret keys $T$ may perfectly forge communication with $C$ (e.g. an execution in which $C$ presents a signature)</td>
</tr>
<tr>
<td>- a weaker adversary may be trusted (e.g. if $T$ has no private keys of $C$, it cannot forge the signature)</td>
</tr>
</tbody>
</table>
Cryptographic protocols have to be extremely simple, otherwise a secure implementation might be quite hard.
Thanks for your attention!

Contact data

1. Miroslaw.Kutylowski@pwr.edu.pl