My Own Little Hilbert’s Program

Sebastian Mödersheim
Danmarks Tekniske Universitet
samo@imm.dtu.dk

Based on joint work with:
Thomas Groß, University of Newcastle
Luca Viganò, Università di Verona
Hanne Riis Nielson & Flemming Nielson, DTU

Séminaire de Cryptologie, 18.10.2012
Type-Flaw Attacks

- Confusion about the interpretation of messages
- Lower-level/implementation problems
- Often not so interesting
- Get in the way in verification of larger systems
- Can we abstract from them?

\[
\begin{array}{ccc}
M & A & B \\
1001101100111100 & 11011011 & 00010010 \\
\text{Encryption} & \text{Decryption} & \text{Intended interpretation of sender} \\
\text{Ciphertext} \\
1001101100111100 & 11011011 & 00010010 \\
\text{Interpretation of receiver} \\
K_{AB}
\end{array}
\]
“Typed model” in symbolic protocol verification

- Different from standard type systems like Hindley-Milner
- Receivers **magically** can check correct format/type of messages
- **Relative soundness** ([Heather et al], [Malladi & Lafourcade], [Arapinis & Duflot])
  
  "If there is an attack, then there is also a well-typed attack."

- **Helpful**: complexity/decidability

Protocol Composition

- **Disjointness conditions**: message parts of the different protocols can be distinguished. [Guttman], [Cortier & Delaune], [Groß & M.]
- Similar requirements & proofs as for typing — **protocol types**

“Typed” Diffie-Hellman

- Agents **magically** can check that received half-keys are of the form \( \exp(g, X) \)
- Soundness result for a certain class of protocols ([M.], [Lynch & Meadows])
Hilbert’s Program

“. . . to recognize our common methods of mathematics in their entirety as consistent.”

Some say this program has failed, but in a brilliant way!
My Hilbert’s Program

“. . . to recognize our common models of security in their entirety as typeable.”
• Relate questions of typing, compositional and algebraic reasoning.
• Good engineering practice anyway: non-atomic message parts with different meaning must be distinguishable.
• Abstract from lower-level details like parsing.

Challenge: prove this program to fail as well!
Outline

1. Typeability
2. Disjointness in Composition
3. Typing for Diffie-Hellman
4. Typeable ASLan
5. Lazy Mobile Intruders
6. Conclusions & Outlook
Outline

1. Typeability
2. Disjointness in Composition
3. Typing for Diffie-Hellman
4. Typeable ASLan
5. Lazy Mobile Intruders
6. Conclusions & Outlook
## Message Model

### Operations

E.g. crypt, scrypt, exp, \([\cdot]_n\), ...  
- Every agent, including intruder, can apply these operations.

Abstracting from parsing problems: \(n\)-ary concatenation \([\cdot]_n\).

### Mappings

E.g. inv, pk, sk, ...  
- Define **atomic terms** as the closure of all constants and variables under the mapping functions.
### Atomic Types
Typed variables can only be substituted with atomic terms of that type.

### Composed Types
Closure of atomic types under operation symbols, e.g.

\[ M : \text{scrypt}(\text{hash}(\text{nonce}, \text{nonce}), [\text{nonce}, \text{nonce}]) \].

Semantics: replace \( M \) by appropriate term with fresh vars.
A Typing Result
similar to [Heather et al] and [Arapiinis & Duflot]

Well-Designed Protocols

- AVISPA IF protocol $P$ without negative facts and conditions
- Interpreted in free algebra
- Type annotation $\Gamma$ for all variables
- $MP(P)$ set of all message patterns of honest agents
- $SMP(P)$ non-atomic subterms of $MP(P)$
- $P$ is well-designed (w.r.t. $\Gamma$): no two elements of $SMP(P)$ that have different composed type can be unified

Theorem

Then if $P$ has an attack, it also has a well-typed attack.

Proof idea: insertion of ill-typed messages never helps the intruder.
The Lazy Intruder

Symbolic/Constraint-based Approach

[Huima], [Amadio & Lugiez], [Millen & Shmatikov], [Chevalier & Vigneron], [Basin & M. & Viganò], [Delaune et al.]...

- Avoid exploration of intruder choices by collecting symbolic constraints $M ⊢ m$.
- Correct and terminating reduction calculus for several algebraic theories.
The Lazy Intruder: Example

Example: PKInit Kerberos

\[ C \rightarrow Auth : [C, \text{crypt}(\text{inv}_{pk_C}, N)] \]
\[ Auth \rightarrow C : \text{crypt}(pk_C, \text{crypt}(\text{inv}_{pk_{Auth}}, K_{temp})), \text{scrypt}(K_{temp}, \ldots) \]

Intruder as dishonest client \( C' \) with initial knowledge
\[ K_0 = \{pk_C, pk_{Auth}, pk_{C'}, \text{inv}_{pk_{C'}}\} : \]

\[ K_1 := K_0 \cup \{[C, \text{crypt}(\text{inv}_{pk_C}, N)]\} \]
\[ K_1 \vdash [X, \text{crypt}(\text{inv}_{pk_X}, Y)] \]

\[ K_2 := K_1 \cup \{\text{crypt}(pk_X, \text{crypt}(\text{inv}_{pk_{Auth}}, K_{temp})), \text{scrypt}(K_{temp}, \ldots)\} \]
\[ K_2 \vdash \text{crypt}(pk_C, \text{crypt}(\text{inv}_{pk_{Auth}}, Z)), \text{scrypt}(Z, \ldots) \]
The Lazy Intruder as a Proof Technique

- If a protocol has an attack, then there are well-formed $M \vdash m$ constraints, such that every solution represents an attack.
- The usual constraint reduction is complete
- For a well-designed protocol, the constraint reduction never performs an ill-typed substitution.
  - Substitutions occur only when unifying two terms — and these terms cannot be variables.
Outline

1 Typeability

2 Disjointness in Composition

3 Typing for Diffie-Hellman

4 Typeable ASLan

5 Lazy Mobile Intruders

6 Conclusions & Outlook
Typing in Compositional Reasoning

Similar results for parallel composition of protocols:

- Let protocols $P_1$ and $P_2$ be are disjoint:
  - No message of $SMP(P_1)$ can be unified with a message of $SMP(P_2)$.
- Protocols have same set of long-term public values
- There are no side-channels such as databases
- **Thm:** $P_1$ and $P_2$ can safely be run in parallel.
  - Proof: lazy intruder never unifies $P_1$ with $P_2$ terms.
- Seen as typing: messages have either type $P_1$ or type $P_2$
Example of Vertical Composition

Is this secure?
An Extended Notion of Disjointness

Definition (Message Patterns Modulo Encryption)

again, everything under appropriate $\alpha$-renaming

- $MP(P)$: message patterns of protocol $P$
- $EMP_0(P) = MP(P)$
- $EMP_{n+1}(P) = \{\text{scrypt}(K_{n+1}, m) \mid m \in EMP_n(P)\}$
- $EMP(P) = \bigcup_{n \in \mathbb{N}} EMP_n(P)$
- $EST(P)$: non-atomic subterms of $EMP(P)$

Protocols must be disjoint from their own encryptions:
- no unifier between patterns of $EMP_i(P)$ and $EMP_j(P)$ for $i \neq j$.

Protocols must be disjoint from each other modulo encryptions:
- no unifier between patterns of $EST(P)$ and $EST(Q)$ for $P \neq Q$
Protocol suite $\mathcal{P}$ that satisfies all conditions so far:

In an arbitrary composition of protocols of $\mathcal{P}$, every non-atomic message part can be attributed to a unique protocol stack.

Gives rise to extended notion of protocol types such as

$$http\langle TLS\rangle\langle IPSec\rangle\langle TLS\rangle$$

**Theorem**

*Every composition of protocols in $\mathcal{P}$ is secure.*
Outline

1. Typeability
2. Disjointness in Composition
3. Typing for Diffie-Hellman
4. Typeable ASLan
5. Lazy Mobile Intruders
6. Conclusions & Outlook
Typing for Diffie-Hellman

Diffie-Hellman and Typeability

Unrealistic Diffie-Hellman

\[ A \rightarrow B : \text{crypt}(\text{inv}_{pq_A}, [B, \exp(g, X)]_2) \]

\[ \ldots \]

B’s message pattern for receiving:

\[ \text{crypt}(\text{inv}_{pq_A}, [B, \exp(g, X)]_2) \]

Actually B cannot check the “format” of the red part.

- This means a significant state space reduction
- Can we justify such a model?
## Diffie-Hellman without Difficulty

### Requirements for our soundness result

- Already part of typeability: half-keys are distinguished from other message parts.
- Fresh exponents for every exchange.
- Exponents only used as:
  - Half-key: \( \exp(g, X) \)
  - Full-key in a symmetric encryption: \( \text{scrypt}(\exp(GX, Y), \cdot) \)
    where \( GX \) received as half-key
- Exponentiation does not occur elsewhere

Captured as additional well-formedness conditions for the lazy intruder.
Theorem

Said class of Diffie-Hellman protocols is typeable.

Proof:

- Modified notion of simple lazy intruder constraints:
  
  Additionally \( \exp(GX, x) \) is considered as simple (will not be reduced) if the constraint introducing \( GX \) is already simple.

- Related restriction on unifying terms
- Proof that resulting lazy intruder is still complete
- ... and does not perform ill-typed substitutions.
Abstract Diffie-Hellman to primitives for half-key (kap) and full-key (kas):

\[ \text{kas}(\text{kap}(X), Y) \approx \text{kas}(\text{kap}(Y), X) \]

- Typeability implies that this model is sound relative to our original Diffie-Hellman model.
- This allows for encoding into free algebra by case distinction. (Similar to [Küsters & Truderung])
- Case distinction can even be avoided: ensure type initiator and type responder in half-key exchange!
Outline

1. Typeability
2. Disjointness in Composition
3. Typing for Diffie-Hellman
4. Typeable ASLan
5. Lazy Mobile Intruders
6. Conclusions & Outlook
ASLan: AVANTSSAR specification language

• Combination of two concepts:
  ★ AVISPA IF: infinite-state transition system, based on set-rewriting
  ★ Horn clauses: immediate evaluations at every state

• Useful for modeling of complex systems (and properties):
  ★ Classic example: intruder deduction in a protocol.
  ★ Integration of workflow with policies
  ★ Participants maintain databases and make internal computations in these databases.
  ★ Dynamic virtualized infrastructure with information flow evaluations.

• Undecidability: Horn clauses alone allow for logic programming
Example

Declarations:

\[
\begin{align*}
mem & : \text{pred}(agent, gid) & \text{own} & : \text{pred}(gid, fid) \\
\text{deputy} & : \text{pred}(agent, agent) & \text{xs} & : \text{pred}(agent, fid) \\
\text{attack} & : \text{pred}() & A, B, a, b & : \text{agent} \\
G, G_1, G_2, g_1, g_2 & : \text{gid} & F, F_1, F_2, f_1, f_2 & : \text{fid}
\end{align*}
\]

Initial State:

\[
\text{mem}(a, g_1) \land \text{mem}(b, g_2) \land \text{own}(g_1, f_1) \land \text{own}(g_2, f_2)
\]

Transition Rules:

\[
\begin{align*}
\text{mem}(A, G_1) \land \neg \exists B : \text{deputy}(A, B) & \Rightarrow \text{mem}(A, G_2) \\
\text{xs}(A, F_1) \land \text{xs}(A, F_2) \land \text{own}(G_1, F_1) \land \text{own}(G_2, F_2) \land G_1 \neq G_2 & \Rightarrow \text{attack}()
\end{align*}
\]

Horn clauses:

\[
\begin{align*}
\text{mem}(A, G) \land \text{own}(G, F) & \Rightarrow \text{xs}(A, F) \\
\text{deputy}(A, B) \land \text{xs}(B, F) & \Rightarrow \text{xs}(A, F)
\end{align*}
\]
Example

\begin{align*}
\text{mem}(a,g1) \\
\text{mem}(b,g2) \\
\text{own}(g1,f1) \\
\text{own}(g2,f2)
\end{align*}
Example

\[ \text{mem}(a, g1) \wedge \neg \exists B : \text{deputy}(A, B) \Rightarrow [G2] \Rightarrow \text{mem}(A, G2) \]
Example

\[ \text{mem}(A, G1) \land \neg \exists B : \text{deputy}(A, B) \Rightarrow [G2] \Rightarrow \text{mem}(A, G2) \]
Example

\[
\text{mem}(A, G1) \land \neg \exists B : \text{deputy}(A, B) \Rightarrow \text{mem}(A, G2)
\]
Example

mem(a, g1)
mem(b, g2)
own(g1, f1)
own(g2, f2)
Example

\[
\begin{align*}
\text{mem}(a,g1) \\
\text{mem}(b,g2) \\
\text{own}(g1,f1) \\
\text{own}(g2,f2) \\
\text{xs}(a,f1) \\
\text{xs}(b,f2)
\end{align*}
\]
Bob assigns a deputy (simplified):

\[
\text{mem}(A, G1) \land \text{mem}(B, G2) \Rightarrow \\
\text{mem}(A, G1) \land \text{mem}(B, G2) \land \text{deputy}(A, B)
\]
Example

\[
\begin{align*}
\text{mem}(a, g1) & \quad \text{mem}(a, g1) \\
\text{mem}(b, g2) & \quad \text{mem}(b, g2) \\
\text{own}(g1, f1) & \quad \text{deputy}(a, b) \\
\text{own}(g2, f2) & \quad \text{own}(g1, f1) \\
\text{xs}(a, f1) & \quad \text{xs}(a, f1) \\
\text{xs}(b, f2) & \quad \text{xs}(a, f2) \\
\end{align*}
\]
\[ xs(A, F1) \land xs(A, F2) \land own(G1, F1) \land own(G2, F2) \land G1 \neq G2 \Rightarrow attack() \]
Symbolic state:

\[ \phi ::= P \]

\[ S \vdash P \]

\[ \neg \exists \vec{X} : s_1 = t_1 \land \ldots \land s_n = t_n \]

\[ X = t \]

\[ \phi \land \psi \]

**Lemma**

*This allows for a finitely branching symbolic transition relation.*
### Symbolic State Satisfiability

<table>
<thead>
<tr>
<th>Constraint type</th>
<th>Satisfiability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intruder deduction constraints</td>
<td>Lazy Intruder</td>
</tr>
<tr>
<td>$\text{ik}(t_1) \land \ldots \land \text{ik}(t_n) \vdash \text{ik}(t)$</td>
<td>(NP-complete)</td>
</tr>
<tr>
<td>Other deduction constraints $S \vdash P$</td>
<td>Undecidable in general</td>
</tr>
<tr>
<td>Negated substitutions</td>
<td>Unification after substituting</td>
</tr>
<tr>
<td>$\neg \exists \vec{X} : s_1 = t_1 \land \ldots \land s_n = t_n$</td>
<td>bound variables with fresh values</td>
</tr>
<tr>
<td>Substitutions $X = t$</td>
<td>(Just bookkeeping)</td>
</tr>
</tbody>
</table>
Typeable ASLan

TASLan

\textit{SMP}: non-atomic subterms of \( ik(t) \) of transition rules.

\begin{definition}
TASLan is ASLan with following requirements/modifications:
\begin{itemize}
  \item \textbf{Classical disjointness}: If \( t_1, t_2 \in \textit{SMP} \) have a unifier, then \( t_1 \) and \( t_2 \) must have the same type.
  \item \textbf{Type untyped} occurs only in intruder deduction.
\end{itemize}
\end{definition}

For instance you \textbf{cannot} have:

\begin{align*}
p(z) \\
\forall N. p(N) &\rightarrow p(s(N))
\end{align*}

For many specifications these restrictions are fine!
Typeable ASLan

Results

Theorem

For TASLan the following problems are decidable:

- Satisfiability of symbolic states
- Reachability of an attack state in $l$ steps (NEXPTIME-complete)

If a symbolic state is satisfiable, then it is satisfiable under a well-typed interpretation.

Proof.

In TASLan:

- Applying Horn clauses does not introduce ill-typed substitutions.
- The lazy intruder never introduces ill-typed substitutions.
- This allows for a convergent evaluation relation for the $S \vdash P$. 
Outline

1. Typeability
2. Disjointness in Composition
3. Typing for Diffie-Hellman
4. Typeable ASLan
5. Lazy Mobile Intruders
6. Conclusions & Outlook
Lazy Mobile Intruders

Problem

Given:

• Platform $C[\cdot]$ that executes potentially malicious code
• Initial intruder knowledge $K_0$
• Attack predicate $attack(S)$

Question: exist code $P$ and state $S$ such that

• $K_0 \vdash P$
  The intruder can generate the code from his initial knowledge
• $attack(S)$
• $C[P] \rightarrow^* S$
  The platform can reach an attack state when executing $P$.

Obviously we cannot search the space of all programs $K_0 \vdash P$!
Lazy Mobile Intruders

Idea: [M. & Nielson & Nielson]

- Generate the code in a demand-driven, lazy way
- Code is initially $\boxed{K}$ for knowledge $K$.
- Determine code step by step by what transitions are possible
- When reading message: augment $K$
- When writing message: $K \vdash x$
- When two intruder processes meet: pool knowledge
- Analyze what locations the intruder code can reach and with what knowledge
- Decision procedure for mobile ambient calculus without replication:
  - ★ Platform can perform only bounded number of steps
  - ★ Intruder code not bounded
Outline

1. Typeability
2. Disjointness in Composition
3. Typing for Diffie-Hellman
4. Typeable ASLan
5. Lazy Mobile Intruders
6. Conclusions & Outlook
Conclusions & Outlook

The **Lazy Intruder** is a versatile idea

- **Protocol Verification**
  - Bounded sessions, several algebraic theories
  - Now even unbounded case... [Guttman et al]
  - Typeable ASLan: transition system + Horn clauses

- **Relative soundness results:**
  - Different notions of typing: messages, protocols, channels
  - Reduce complex verification problems to smaller ones in algebraic and compositional reasoning
  - Proofs are similar, abusing the lazy intruder as an argument
  - Exploiting what is good engineering practice anyway!

- **Lazy invention of intruder-generated code**

A lot left to do:

- Less restrictive assumptions, more algebraic theories
- Broader scope of channel properties for compositional reasoning
- Privacy/unlinkeability