Cryptography does not imply security
Protocol

\[\{ ni, I \}pk(R)\]

\[\{ ni, nr \}pk(I)\]

\[\{ nr \}pk(R)\]

ni and nr are secret

ni and nr are secret
Protocol role

role I
{
  fresh ni: Nonce;
  var nr: Nonce;
  send_1(I,R, {ni,I}pk(R));
  recv_2(R,I, {ni,nr}pk(I));
  send_3(I,R, {nr}pk(R));
  claim(I,Secret,ni);
  claim(I,Secret,nr);
}
Scyther

• Start scyther
• Load (CTRL-O or menu)
  ns3.spdl
• Verify claims (top left menu)
Protocol fix

\{ \text{n}_i, \text{l} \}\text{pk}(\text{R})

\{ \text{n}_i, \text{n}_r \}\text{pk}(\text{l})

\{ \text{n}_r \}\text{pk}(\text{R})

\text{ni and nr are secret}

\text{Fresh n}_i

\text{Fresh n}_r

\text{ni and nr are secret}
Exercise

- “scyther-lab.pdf” in your files/directory

- Claim event guarantees (if correct):
  If an honest agent performs the role up to the claim, trying to communicate with honest peers, then
  - [secret]
    “the message in the claim is secret”
  - [alive]
    “the honest peer(s) performed some action”
  - [nisynch]
    “everything happened as expected”
    (Expert note: cf. “matching histories” with time ordering)
Exercise review
What is the problem?
99 problems...

- 2003: PKCS#11 crypto API attacks
- 2008: Google single-sign on protocol (SAML) attack
- 2009: TLS renegotiation attack
- 2012: ISO 9798 authentication standard attacks
- 2014: TLS Triple handshake attack
- 2014: ISO 11770 key exchange standard attacks
- 2015: Freak attack on TLS
- Etc etc

Result: **insecure**
- No problems with cryptographic primitives
- No problems with probabilities
Secure session?
Internet Key Exchange (IKE, in IPv6)

“IKE is fairly complicated; to fully understand it, it’s helpful to possess *multiple advanced degrees in mathematics and cryptography* and to have *copious amounts of spare time* to read many detailed yet highly valuable resources.”

Microsoft TechNet: *How IPsec works*


(Retrieved in 2011 and again on August 29, 2016)
Example IKE exchange

1. A → B : HDR₁, SA, gₓₐ, Nₐ, IDₐ
2. B → A : HDR₂, SA’, gₓₜ, Nₐ, IDₐ,
   \{prf₃(gₓₜ, gₓₐ, CKₜₜ, CKₐₐ, IDₜ)\}sk(B)
3. A → B : HDR₃, \{prf₃(gₓₛ, gₓₜ, CKₛₜ, CKₜₜ, IDₛ)\}sk(A)

where K = prf₁(Nₐ, Nₜ)(gₓₙₓₜ).
IKEv1 Aggressive Mode with digital signatures

1. $A \rightarrow B$: HDR$_1$, SA, $g^x$, $N_A$, $ID_A$
2. $B \rightarrow A$: HDR$_2$, SA, $g^y$, $N_B$, $ID_B$, $\{prf_K(g^y, g^x, CKY_B, CKY_A, ID_B)\}_{sk(B)}$
3. $A \rightarrow B$: HDR$_3$, $\{prf_K(g^y, g^x, CKY_A, CKY_B, ID_A)\}_{sk(A)}$

IKEv1 Main Mode with digital signatures
IKEv1 Aggressive Mode with digital signatures

IKEv1 Main Mode with digital signatures

IKEv1 Aggressive Mode with Pre-shared keys

IKEv1 Main Mode with Pre-shared keys

IKEv1 Aggressive Mode with Public keys

IKEv1 Main Mode with Public keys

IKEv1 Aggressive Mode with Public keys (2)

IKEv1 Main Mode with Public keys (2)

Note: some minor variants omitted!
<table>
<thead>
<tr>
<th>Phase 1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IKEv1 Aggressive Mode with digital signatures</td>
<td>IKEv1 Main Mode with digital signatures</td>
</tr>
<tr>
<td>IKEv1 Aggressive Mode with Pre-shared keys</td>
<td>IKEv1 Main Mode with Pre-shared keys</td>
</tr>
<tr>
<td>IKEv1 Aggressive Mode with Public keys</td>
<td>IKEv1 Main Mode with Public keys</td>
</tr>
<tr>
<td>IKEv1 Aggressive Mode with Public keys (2)</td>
<td>IKEv1 Main Mode with Public keys (2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IKEv1 Quick Mode</td>
<td>IKEv1 Quick Mode without PFS</td>
</tr>
<tr>
<td>IKEv1 Quick Mode without Identity</td>
<td></td>
</tr>
</tbody>
</table>

Note: some minor variants omitted!
## IKEv1

### Phase 1
- IKEv1 Aggressive Mode with digital signatures
- IKEv1 Main Mode with digital signatures
- IKEv1 Aggressive Mode with Pre-shared keys
- IKEv1 Main Mode with Pre-shared keys
- IKEv1 Aggressive Mode with Public keys
- IKEv1 Main Mode with Public keys
- IKEv1 Aggressive Mode with Public keys (2)
- IKEv1 Main Mode with Public keys (2)

### Phase 2
- IKEv1 Quick Mode
- IKEv1 Quick Mode without PFS
- IKEv1 Quick Mode without Identity

## IKEv2

### Phase 1
- IKEv2 SIG
- IKEv2 SIG noid
- IKEv2 MAC
- IKEv2 MAC noid
- IKEv2 EAP
- IKEv2 EAP noid
- IKEv2 SIG/MAC asymmetric variants
- IKEv2 SIG/MAC asymmetric variants

### Phase 2
- IKEv2 child mode
- IKEv2 child mode without PFS

Note: some minor variants omitted!
“The state-of-the-art is able to handle [...] protocols of realistic, but limited, complexity. A **fully automatic analysis** of the Internet Key Exchange Protocol, with all its variations, would lead to state explosion and is **outside of the current state-of-the-art.**”

David Basin, *Cas Cremers*, Catherine Meadows

“**Model Checking Security Protocols**“ [Draft]
Chapter for the Handbook of Model Checking, to appear.

(Retrieved May 1, 2011)
Combining protocols

Consider the following bike parking:

Note: There are more bikes than the guard can count (>10)
Combining protocols

Bike chain around rack and front wheel.

If everybody does this, a thief can't steal a whole bike, and bike away!
Combining protocols

Bike chain around rack and **frame**.

If *everybody* does this, a thief can't steal a whole bike, and bike away!
Combining protocols

If people mix the methods, a thief can steal a complete bike! **Conclusion:** mixing several correct (secure) security protocols together can be incorrect (insecure)!
Problem: security does not compose

Theoretical result 1:

“chosen protocol attack” (Kelsey, Schneier, Wagner, 1997)

Given a secure protocol, there exists a second one (also secure) such that their composition is not secure.

Protocol 1: \[ A \rightarrow B : \{ \text{challenge, B } \}^{kab} \]

\[ B \rightarrow A : \text{challenge} \]

Protocol 2: \[ A \rightarrow B : \{ \text{my_secret, B } \}^{kab} \]

Attack: secret message from Protocol 2 redirected into Prot. 1
Real-life protocols

Interesting protocol sets

Trivially composing
(Disjoint encr.)

Protocol sets that Safely compose

All protocol sets
Modern adversary/threat models

- Adversary can
  - learn long-term keys,
  - learn the randomness generated in sessions,
  - learn session keys
  - learn (part of) the session state

- Security guarantee holds for all clean sessions
  - A complex condition that involves:
    - All other sessions
    - Checking partial authentication
    - Temporal ordering of events
Can tools help out?
Scyther (Cremers, 2006)

- Focusses on event structures
- Does **not use abstraction**
  - Never finds ``false'' attacks
- Input language: domain-specific language (SPDL)
  - Linear role scripts
Basis: Dolev Yao adversary model

- Models an active intruder with full network control and perfect recall
- Idealized black-box cryptography

\[
\begin{align*}
t \in M & \quad \frac{M \vdash t_1}{\frac{M \vdash (t_1, t_2)}{M \vdash t}} \quad \frac{M \vdash (t_1, t_2)}{M \vdash t_1} \quad \frac{M \vdash (t_1, t_2)}{M \vdash t_2} \\
M \vdash t & \quad \frac{M \vdash h(t)}{M \vdash \{t_1\}_{t_2}} \quad \frac{M \vdash \{t_1\}_{t_2}}{M \vdash t_{t_1^{-1}}}
\end{align*}
\]

Successful: interesting theory and powerful tools
Terms, roles, and protocols

- **Terms**: operators for constructing cryptographic messages

  \[ \text{Term} ::= \text{Agent} \mid \text{Fresh} \mid \text{Var} \mid (\text{Term}, \text{Term}) \mid \{\text{Term}\}_{\text{Term}} \mid \ldots \]

- **Roles**: sequences of **agent events**

  \[ \text{AgentEvent} ::= \text{create}(\text{Role}, \text{Agent}) \mid \text{send}(\text{Agent}, \text{Agent}, \text{Term}) \mid \text{recv}(\text{Agent}, \text{Agent}, \text{Term}) \]

- **Example**

  \[
  I \rightarrow R : \quad \{I, K\}_{K_{IR}} \\
  R \rightarrow I : \quad \{R, M\}_K
  \]

  \[
  P(I) = [ \quad \text{send}(I, R, \{I, K\}_{k(I,R)}); \quad \text{recv}(R, I, \{R, y\}_K) \quad ]
  \]

  \[
  P(R) = [ \quad \text{recv}(I, R, \{I, x\}_{k(I,R)}); \quad \text{send}(R, I, \{R, M\}_x) \quad ]
  \]
Threads

- A **thread** is a role instance (local session)
  - No limit to number of threads
  - Each thread assigned a unique identifier from the set TID.
  - We **instantiate names** and **syntactically bind** fresh **values** and **variables** to their owning thread, e.g. K#1, y#1

\[
P(I) = \begin{array}{c}
generate(\{K\}); \\
send(I, R, \{I, K\}_k(I, R)); \\
recv(R, I, \{R, y\}_k) \\
generate(\{K#1\}); \\
send(A, B, \{A, K#1\}_k(A, B)); \\
recv(B, A, \{B, y#1\}_k) \\
\end{array}
\]

- For currently active threads, we store the remaining sequence of steps in a **thread pool** \( th : TID \rightarrow \text{AgentEvent}^* \)
Core symbolic model

- Goal: to define a transition system with all possible behaviours
Core symbolic model

- Goal: to define a transition system with all possible behaviours

Respnder Dave receives a message that was produced by the adversary.
Core symbolic model

- Goal: to define a transition system with all possible behaviours

Respnder Dave receives a message that was produced by the adversary

Alice starts a new thread as initiator, trying to talk to Gavin
Core symbolic model
(slightly simplified)

- **State** \((\text{tr,IK,th})\)
  - \(\text{tr}\) : trace of events that have occurred
  - \(\text{IK}\) : “intruder knowledge” of adversary, initially \(\text{IK}_0\)
  - \(\text{th}\) : thread pool, mapping thread identifiers to remaining steps
- **Transition system** modeling agents' threads and adversary

\[
\begin{align*}
\text{th}(\text{tid}) &= \langle \text{send}(m) \rangle^l \quad \text{[send]} \\
(\text{tr,IK,th}) &\rightarrow (\text{tr}^\langle (\text{tid}, \text{send}(m)) \rangle, \text{IK} \cup \{m\}, \text{th}[l \leftarrow \text{tid}])
\end{align*}
\]

\[
\begin{align*}
\text{th}(\text{tid}) &= \langle \text{recv}(\text{pt}) \rangle^l \quad \text{IK} \vdash \sigma(\text{pt}) \quad \text{dom}(\sigma) = \text{FV}(\text{pt}) \quad \text{[recv]} \\
(\text{tr,IK,th}) &\rightarrow (\text{tr}^\langle (\text{tid}, \text{recv}(\sigma(\text{pt}))) \rangle, \text{IK}, \text{th}[\sigma(l) \leftarrow \text{tid}])
\end{align*}
\]

Example of reachable state:

\[
\left( \langle (1, \text{send}(A, B, n\#1)) \rangle, \text{IK}_0 \cup \{n\#1\}, \{1 \mapsto \langle \rangle, 2 \mapsto \langle \text{recv}(A, B, X\#2) \rangle \} \right)
\]
Reasoning about protocol semantics (TS)

• General complexity
  – **Reachability** properties are **undecidable**, e.g. secrecy
    (Durgin, Lincoln, Mitchell, Scedrov 1999)
  – **NP-hard**, even when number of sessions is bounded
    (Rusinowitch, Turuani, 1999)

• Scyther tool often successful in protocol analysis
<table>
<thead>
<tr>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>

2008
Scyther pros and cons

- **Pros**
  - Unbounded analysis by backwards search
    - no bound on the number of possible threads in attacks
  - Fast, push-button
  - Many case studies
  - Support for different adversary models

- **Cons**
  - Linear role scripts
    - No if/then
    - No loops within protocol
  - No good support for equational theories
  - No mutable global state
  - Fixed set of security properties
The Tamarin Prover
The Tamarin Prover

Family of small monkeys in South America
Choice: Emperor Tamarin
Important: Not near extinction

Joint work with:
Tamarin prover: History

• Idea: generalize Scyther's approach
  – Better support for Diffie-Hellman
  – Loops, branches
  – Property specification

• From vague idea to theory to tool between 2008 and 2012
  – Simon and Benedikt: vast majority of the development
  – Cedric Staub worked on the GUI
  – Many people involved in models and tool extensions
  – Several person years of work
The ISO/IEC 9798 Standard

• Entity Authentication Mechanisms

• **18 base protocols**
  – Symmetric-key encryption, Digital signatures, Cryptographic check functions
  – Unilateral or Mutual authentication
  – Additional protocols with TTP

• **Further variants** from optional fields
The ISO/IEC 9798 Standard

• History
  • Active development and updates since 1991
  • Blueprints for protocol design
  • Basis for ISO 11770 (Key Exchange) and NIST FIPS 196
  • Mandated by other standards
    - e.g. European Banking Commission's smart card standards

• Intended properties
  • Entity authentication?
  • E.g. Resistant to reflection attacks
  • Encrypted/signed payloads?
ISO 9798-2-5

Trusted Third Party

\[ P \]

\[ TVP_A, l_B, Text_1 \]

Token\(_{PA} = Text_4, \]
\[ \{ TVP_A, kab, l_B, Text_3 \}^s_{KAP} \]
\[ \{ TN_P, kab, l_A, Text_2 \}^s_{KBP} \]

Token\(_{PA} \)

\[ Token_{AB} = Text_6, \]
\[ \{ TN_P, kab, l_A, Text_2 \}^s_{KBP} \]
\[ \{ TN_A, l_B, Text_5 \}^s_{kab} \]

Token\(_{AB} \)

\[ Token_{BA} = Text_8, \]
\[ \{ TN_B, l_A, Text_7 \}^s_{kab} \]

Token\(_{BA} \)
Analysis

• Request by CryptRec to evaluate standard
  - Cryptography Research and Evaluation Committees
  - Funded by the Japanese government
  - Part of long-running program to evaluate cryptographic mechanisms

• Confirmation expected
  • Standard has been improved since 1994
  • Multiple previous analysis
Tools used

Scyther
Symbolic analysis of security protocols
- Falsification (attack finding)
- Unbounded verification

Scyther-proof
- Embedding of protocol semantics and protocol-independent invariant in the Isabelle/HOL theorem prover
- Algorithm similar to Scyther that outputs proof script for Isabelle/HOL
- Independent verifiability
Results

- No strong authentication properties
  Aliveness < Agreement < Synchronisation
- Under some conditions no authentication

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Violated property</th>
<th>Assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>9798-2-3</td>
<td>A Agreement(B, TNB, Text3)</td>
<td></td>
</tr>
<tr>
<td>9798-2-3</td>
<td>B Agreement(A, TNA, Text1)</td>
<td></td>
</tr>
<tr>
<td>9798-2-3-udkey</td>
<td>A Agreement(B, TNB, Text3)</td>
<td></td>
</tr>
<tr>
<td>9798-2-3-udkey</td>
<td>B Agreement(A, TNA, Text1)</td>
<td></td>
</tr>
<tr>
<td>9798-2-5</td>
<td>A Alive</td>
<td>Alice-talks-to-Alice</td>
</tr>
<tr>
<td>9798-2-5</td>
<td>B Alive</td>
<td></td>
</tr>
<tr>
<td>9798-2-6</td>
<td>A Alive</td>
<td></td>
</tr>
<tr>
<td>9798-2-6</td>
<td>B Alive</td>
<td></td>
</tr>
<tr>
<td>9798-3-3</td>
<td>A Agreement(B, TNB, Text3)</td>
<td></td>
</tr>
<tr>
<td>9798-3-3</td>
<td>B Agreement(A, TNA, Text1)</td>
<td></td>
</tr>
<tr>
<td>9798-3-7-1</td>
<td>A Agreement(B, Ra, Rb, Text8)</td>
<td>Type-flaw</td>
</tr>
<tr>
<td>9798-4-3</td>
<td>A Agreement(B, TNb, Text3)</td>
<td></td>
</tr>
<tr>
<td>9798-4-3</td>
<td>B Agreement(A, TNa, Text1)</td>
<td></td>
</tr>
<tr>
<td>9798-4-3-udkey</td>
<td>A Agreement(B, TNb, Text3)</td>
<td></td>
</tr>
<tr>
<td>9798-4-3-udkey</td>
<td>B Agreement(A, TNa, Text1)</td>
<td></td>
</tr>
</tbody>
</table>
Thread 1
role P
executed by Pete
assumes Alice in role A
assumes Bob in role B

Thread 2
role A
executed by Pete
assumes Alice in role P
assumes Bob in role B

Thread 3
role B
executed by Bob
assumes Alice in role A
assumes Pete in role P

Mirrored assumptions on A and P players

$K_{AP} == K_{PA}$ - mismatch not detected!

Thread 2 does not decrypt this and therefore does not detect that it is not $K_{BA}$ and $I_{Pete}$

Message does not contain anything of A/P assumptions

Alice

$Token_{PA} = Text_4, \{ TVP_A, k, I_{Bob}, Text_3 \}^{K_{AP}}$}

$Token_{AB} = Text_6, \{ TN_P, k, I_{Alice}, Text_2 \}^{s}_{K_{BP}}$

$\{ TN_A, I_{Bob}, Text_5 \}^{s}_{k}$

$Token_{BA}$

Alice Lives!
Root Causes of the Problems

- Message format is **consistent** and minimal
  - Good design individually, but leads to possible confusion between different messages

- **No type information** for fields
  - Combined with above, can lead to type flaw attacks

- Identity of **one agent** always included to break symmetry of shared keys
  - Great but doesn't work for three parties
Repairing ISO/IEC 9798

• We proposed fixes and machine-checked correctness proofs
  • Fixes do not require additional cryptography

• Scyther-proof generates proof scripts for Isabelle-HOL
  • Minor extension over original [CSF2011] developed for bidirectional keys

• Proofs even guarantee correctness when executing all ISO 9798 protocols in parallel
  • Exclude multi-protocol attacks
Effort

• Modeling effort: a couple of weeks
  • Abstraction level of standard close to formal models
  • Some iteration inevitable after initial analysis with scyther

• Generating proof scripts using Scyther-proof
  • 20 seconds

• Checking correctness in Isabelle/HOL
  • 3 hours (correctness for all protocols in parallel)
ISO/IEC 9798: Conclusions

• Improving the standard
  – Old version: only weak authentication, sometimes none
  – Succesful interaction between researchers and standardization committee:
  – New version of the standard has been released which guarantees strong authentication (synchronisation)
  – Machine-checked symbolic proofs of standard

• We later similarly tackled ISO/IEC 11770
Scyther vs Tamarin
Tamarin: model

- **Term algebra**
  - \( \text{enc}(\_\_), \text{dec}(\_\_), \text{h}(\_\_), \_\_\_\_\_\_\_\_\, \_\_\_\_\_\_\_\_\, 1, \ldots \)

- **Equational theory**
  - \( \text{dec}(\text{enc}(m,k),k) = \mathbb{E} m, \)
  - \( (x^y)^z = \mathbb{E} x^{(y*z)}, \)
  - \( (x^{-1})^{-1} = \mathbb{E} x, \ldots \)

- **Facts**
  - \( F(t1,...,tn) \)

- **Transition system**
  - State: multiset of facts
  - Rules: \( l \rightarrow [ a ] \rightarrow r \)

- **Tamarin-specific**
  - Built-in Dolev-Yao attacker rules
    - \( \text{In}(\_), \text{Out}(\_), K(\_) \)
  - Special **Fresh** rule:
    - \([ [] --[]--> [ \text{Fr}(x) ] \)
      - With additional constraints on systems such that \( x \) unique
Semantics

- **Transition relation**

  \[ S \rightarrow [a] \rightarrow ((S \setminus \# l) \cup \# r) \]

  where \( l \rightarrow [a] \rightarrow r \) is a ground instance of a rule and \( l \subseteq \# S \)

- **Executions**

  \[ \text{Exec}( R) = \{ \emptyset \rightarrow [a_1] \rightarrow \ldots \rightarrow [a_n] \rightarrow S_n \mid \forall n . \text{Fr}(n) \text{ appears only once on rhs} \} \]

- **Traces**

  \[ \text{Traces}( R) = \{ [a_1, \ldots, a_n] \mid \emptyset \rightarrow [a_1] \rightarrow \ldots \rightarrow [a_n] \rightarrow S_n \in \text{Exec}( R) \} \]
Tamarin tackles complex interaction with adversary

Your protocol modeled with rewrite rules

DY-style adversary a.k.a.
The network
The Naxos protocol

\[ \text{Naxos} \]

**I**

fresh \( x \)

\( x_2 = h_1(x, i) \)

\( g^{x_2} \) \[\xrightarrow{\text{fresh } y}\]

\( g^{y_2} \)

**R**

\( y_2 = h_1(y, r) \)

private keys: \( i, r \)

public keys: \( g^i, g^r \)

\[ K = h_2( g^{i \cdot y_2}, g^{r \cdot x_2}, g^{x_2 \cdot y_2}, I, R ) \]
rule generate_ltk:
  let pkI = 'g'^~i
  in
  [ Fr(~i) ]
  -->
  [ Ltk( $I, ~i ) ]

rule Init_1:
  let x2 = h1(<~x, ~i >)
    m1 = 'g'^x2
  in
  [ Fr( ~x ), Ltk( $I, ~i ) ]
  -->
  [ Init_1( ~x, $I, $R, ~i, m1 ), Out( m1 ) ]

rule Init_2:
  [ Init_1( ~x, $I, $R, ~i, m1 ), In( m2 ) ]
  -->
  []
Property specification
Property specification

• 2-sorted \((\text{temp}, \text{msg})\) first order logic interpreted over a trace

  - False
  - Equality \(m_1 =_E m_2\)
  - Timepoint ordering \(#t1 < #t2\)
  - Timepoint equality \(#t1 = #t2\)
  - Action at timepoint \(#t\) \(A@#t\)
Property specification

• Rules:
  - \( I \rightarrow [ a ] \rightarrow r \)
  - Instantiated actions stored as (action) trace
    • Additionally: adversary knows facts: \( K() \)

rule Init_2:
  let pkR = 'g'\(^{r}\),
  x2 = h1(< ~x, ~i >),
  kI = h2(< m2\(^{i}\), pkR\(^{x2}\), m2\(^{x2}\), $I, $R >)
  in
  [ Init_1( ~x, $I, $R, ~i , m1), In( m2 ) ]
  --[ Accept(~x, $I, $R, kI) ]-->
  []

Lemma key_secret:
'')(All #t Test A B k. Accept(Test,A,B,k)@t \Rightarrow Not (Ex #t2. K(k)@t2 ))''
Advanced property specification
eCK security model for key exchange

- Adversary can
  - learn long-term keys,
  - learn the randomness generated in sessions,
  - learn session keys

- But only as long as the Test session is clean:
  - No reveal of session key of Test session or its matching session, and
  - No reveal of randomness of Test session as well as the long-term key of the actor, and
  - If there exists a matching session, then something is disallowed...
  - If there is no matching session, then something else...
Lemma eCK_key_secrecy:
"(All #t1 #t2 Test A B k. Accept(Test, A, B, k) @ t1
& K(k) @ t2 =>

( (Ex #t3. SesskRev( Test ) @ t3 )
| (Ex MatchingSession #t3 #t4 ms.
  ( Sid(MatchingSession, ms) @ t3
  & Match(Test, ms) @ t4)
  & (Ex #t5. SesskRev(MatchingSession) @ t5 ))

| [...]
"
end
Demo
Tamarin: Selected case studies

- Key exchange protocols
  - Naxos
  - Signed DH
  - KEA+
  - UM
  - Tsx
  - TLS handshake
- Group protocols
  - GDH
  - TAK
  - (Sig)Joux
  - STR
- ID-based AKE
  - RYY
  - Scott
  - Chen-Kudla
- Protocols with loops
  - TESLA1
  - TESLA2
- Non-monotonic global state
  - Keyserver
  - Envelope
  - Exclusive secrets
  - Contract signing
  - Security device
  - YubiKey
  - YubiHSM
- PKI with strong guarantees
  - ARPKI (also global state)
- Transparency
  - KUD/DECIM (also global state)
„Nice, but I ♥ Pi calculus (only)“
SAPIC

- Stateful applied Pi calculus + tool
  - Steve Kremer & Robert Künnemann

- Compiles to Tamarin input

\[ P_{\text{Yubikey}} = \]
\[ \nu k; \nu \text{pid}; \nu \text{secid}; \]
\[ \text{insert } \langle \text{Server}, \text{pid} \rangle, \]
\[ \langle \text{secid}, k, \text{‘one’} \rangle; \]
\[ \text{insert } \langle \text{‘Yubikey’}, \text{pid} \rangle, \text{‘one’}; \]
\[ \text{out}(\text{pid}); \]
\[ !P_{\text{ButtonPress}} \]

\[ P_{\text{ButtonPress}} = \]
\[ \text{lock } \langle \text{‘Yubikey’}, \text{pid} \rangle; \]
\[ \text{lookup } \langle \text{‘Yubikey’}, \text{pid} \rangle \text{ as } tc \text{ in } \]
\[ \text{insert } \langle \text{‘Yubikey’}, \text{pid} \rangle, \text{tc} + \text{‘one’}; \]
\[ \nu \text{nonce}; \nu \text{npr}; \]
\[ \text{event } \text{YubiPress}(\text{pid}, \text{secid}, k, tc); \]
\[ \text{out}(\langle \text{pid}, \text{nonce}, \text{senc}(\langle \text{secid}, \text{tc}, \text{npr} \rangle, k) \rangle); \]
\[ \text{unlock } \langle \text{‘Yubikey’}, \text{pid} \rangle \]
Tamarin summary

• We can now deal with:
  – Any number of instances, even with loops and mutable global state
  – Complex protocol details and property specifications
  – Some support for observational (trace) equivalence (2016)
  – But still much left to be handled and automated

• The Tamarin prover is freely available
  – Manual (PDF and website)
  – Theses Simon Meier & Benedikt Schmidt
  – Development on github
Internet Security
Overview

- Case study: TLS 1.3
  - What is it?
  - Our analysis approach
  - Some details
  - Results

- Wrap up
These all implement the **TLS** protocol: **Transport Layer Security**

previously known as SSL;
also the 'S' in 'https';
a.k.a. the green lock

**The purpose of TLS:**
To provide a secure channel to transfer messages
Security of TLS over time

- SSL 2.0
- Bleichenbacher
- 1995
- SSL 3.0
- Vaudenay
- 1996
- TLS 1.0
- Cribbs et al.
- 1999
- TLS 1.1
- Bellare et al.
- 2002
- TLS 1.2
- Belding et al.
- 2005
- Renegotiation
- Paepcke and Bohacek
- 2008
- CRIME
- Maviton and Vaudenay
- 2009
- BEAST
- Paepcke and Bohacek
- 2011
- Lucky Thirteen
- AlFardan et al.
- 2012
- RC4 attacks
- Mueller et al.
- 2013
- Heartbleed
- Neuman et al.
- 2014
- POODLE
- Cremers et al.
- 2015
- Logjam
- Shades et al.
- 2016
- SLOTH
- Jager et al.
TLS development

• Currently under development: **TLS 1.3**
  – Led by the **Internet Engineering Task Force (IETF)**
  – Public mailing list discussions
  – Long, complex process
TLS 1.3

(a) Initial (EC)DHE handshake

(b) 0-RTT handshake

(c) PSK-resumption handshake (+PSK-DHE)
What we did (nutshell)

• Collaboration with Royal Holloway
  – Cas with Marko Horvat, Sam Scott, and Thyla van der Merwe

• We built a symbolic model of the TLS 1.3 specification currently under development (draft 10)

• We wanted to verify the **core** properties of TLS 1.3 as an authenticated key exchange protocol
  – secrecy of session keys
  – unilateral (mutual) authentication

• We found a potential attack – disclosed this to the IETF TLS WG
TLS 1.3 and Tamarin

- We built our model for use in the Tamarin prover
  - Reasons:
    - Supports loops and branches well
    - Good symbolic Diffie-Hellman support
Step 1: Building a model
Step 1: Building a model
Step 1: Building a model
Step 1: Building a model

ClientHello

Receive ServerHello/Finished + Send ClientFinished

Client authentication
Step 1: Building a model

ClientHello

Receive ServerHello/Finished + Send ClientFinished

Client authentication
Step 1: Building a model

ClientHello

Receive ServerHello/Finished + Send ClientFinished

Client authentication
Step 1: Building a model

Client Hello

Receive ServerHello/Finished + Send ClientFinished

Client authentication
Step 1: Building a model

Client Hello

Receive ServerHello/Finished + Send ClientFinished

Client authentication
Step 1: Building a model

```
ClientHello

Receive ServerHello/Finished + Send ClientFinished

Client authentication
```
Step 1: Building a model
Step 1: Building a model

rule C_1:
let

// Default C1 values
tid = ~nc

// Client Hello
C = $C
nc = ~nc
pc = $pc
S = $S

// Client Key Share
ga = 'g'^~a

messages = <nc, pc, ga>
in
[ Fr(nc)
, Fr(~a)
]
--[ C1(tid)
, Start(tid, C, 'client')
, Running(C, S, 'client', nc)
, DH(C, ~a)
]->
[ St_C_1_init(tid, C, nc, pc, S, ~a, messages, 'no_auth')
, Out(<C, nc, pc, ga>)
]
Step 1: Building a model
Step 2: Encoding security properties

- TLS 1.3 goals include
  - unilateral **authentication** of the server (mandatory)
  - mutual **authentication** (optional)
  - **confidentiality** and **perfect forward secrecy** of session keys
  - **integrity** of handshake messages
Step 2: Encoding security properties

secret_session_keys:
(1) “All actor peer role k #i.
(2) SessionKey(actor, peer, role, <k, 'authenticated'>)@i
(3) & not ( (Ex #r. RevLtk(peer)@r & #r < #i)
    | (Ex #r. RevLtk(actor)@r & #r < #i))
(4) ==> not Ex #j. KU(k)@j“

• This says...
  - For all possible values of variables on the first line (1)
  - if key k is accepted at time point i (2), and
  - the adversary has not revealed the long term keys of the actor or the peer before the key is accepted (3)
  - then the adversary cannot derive the key (4)

Want to show that this holds for all combinations of client, server, and adversary behaviours – ALL traces!
Step 3: Proving security properties

What can the adversary do?

What can the adversary do?

and so on...

eventually will boil down to needing to break DH

SessionKey(…)

C2_No_Auth

S2

C2_Auth

S2_Auth
Step 3: Proving security properties

• Not a straightforward application of Tamarin
  – several person-months of work
  – specification a moving target
  – updating takes time, can be error-prone

• Need intimate knowledge of the protocol – high degree of interaction with the tool in some cases
  – Not auto-provable
  – We have 45 auxiliary lemmas
Step 3: Proving security properties

- We verified the core properties of TLS 1.3 draft 10 as an authenticated key exchange protocol:
  - Secrecy of session keys
    - holds for both client and server
    - forward secrecy
  - Mutual authentication
Attacking client authentication (revision 10+)

Analysis:
Tamarin finds an attack!
Attacking client authentication

Alice
(Client)

Charlie
(evil.com)
Attacking client authentication

Alice (Client) → Connect to evil.com → Charlie (evil.com)
Attacking client authentication

Alice (Client) \quad \text{Connect to evil.com} \quad \text{...} \quad \text{Establish PSK} \quad \text{Charlie (evil.com)}
Attacking client authentication

Alice (Client) → Connect to evil.com → ... → Establish PSK → PSK₁ → Charlie (evil.com)
Attacking client authentication

Alice (Client) → Connect to evil.com → Charlie (evil.com) → Establish PSK → PSK₁ → Bob (mybank.com)
Attacking client authentication

Alice (Client)

Connect to evil.com

... 

Establish PSK

PSK₁

Charlie (evil.com)

PSK₁

Connect to mybank.com

... 

Bob (mybank.com)
Attacking client authentication

Alice (Client)  PSK₁  Connect to evil.com  Charlie (evil.com)  PSK₁  PSK₂  Connect to mybank.com  Bob (mybank.com)  PSK₂

Establish PSK  ...  Establish PSK  ...
Attacking client authentication

Alice (Client)  Connect to evil.com  ...  Establish PSK  PSK₁

Charlie (evil.com)  PSK₁ ≠ PSK₂

Connect to mybank.com  ...  Establish PSK  PSK₂

Bob (mybank.com)
Attacking client authentication

Alice (Client)

PSK₁

Session resumption

client_random = nc

Charlie (evil.com)

PSK₁

PSK₂

Bob (mybank.com)

PSK₂

session_hash = H(nc)
Attacking client authentication

Alice (Client)

PSK₁

Session resumption

client_random = nc

Charlie (evil.com)

PSK₁  PSK₂

Session resumption

client_random = nc

Bob (mybank.com)

PSK₂

session_hash = \( H(nc) \)

session_hash = \( H(nc) \)

Attacking client authentication

\[
\begin{align*}
\text{Alice (Client)} & \quad \text{Session resumption} \quad \text{Charlie (evil.com)} \\
\text{PSK}_1 \quad \text{client_random} = nc & \quad \text{PSK}_1 \quad \text{PSK}_2 \\
& \quad \text{Session resumption} \quad \text{Session resumption} \\
& \quad \text{client_random} = nc \quad \text{server_random} = ns \\
& \quad \text{Bob (mybank.com)} \\
\text{session_hash} = H(nc) & \quad \text{session_hash} = H(nc \ ns)
\end{align*}
\]
Attacking client authentication

Alice (Client)

Session resumption

client_random = nc

server_random = ns

PSK₁

Charlie (evil.com)

PSK₁

Bob (mybank.com)

Session resumption

client_random = nc

server_random = ns

PSK₂

PSK₂

session_hash = \( H(nc \ ns \ ... \) )

session_hash = \( H(nc \ ns \ ... \) )
Attacking client authentication

Alice (Client) $\rightarrow$ Charlie (evil.com) $\rightarrow$ Bob (mybank.com)

ClientFinished1

Keys derived from PSK1

PSK1 $\rightarrow$ PSK1 $\rightarrow$ PSK2

\[ \text{session\_hash} = H(nc\ ns\ \ldots) \]

\[ \text{session\_hash} = H(nc\ ns\ \ldots) \]
Attacking client authentication

Alice (Client)

Charlie (evil.com)

Bob (mybank.com)

PSK₁ → Client Finished₁ → PSK₁

Keys derived from PSK₁

PSK₁ → Client Finished₂ → PSK₂

Keys derived from PSK₂

\[ \text{session\_hash} = H(nc \ ns \ ...) \]

\[ \text{session\_hash} = H(nc \ ns \ ...) \]
Attacking client authentication

Alice (Client)

Charlie (evil.com)

Bob (mybank.com)

Request authentication

\[
\text{session\_hash} = H(nc\ ns\ \ldots) \quad \quad \text{session\_hash} = H(nc\ ns\ \ldots)
\]
Attacking client authentication

\[ \text{session\_hash} = H(nc\ ns\ \ldots) \]

\[ \text{session\_hash} = H(nc\ ns\ \ldots) \]
Attacking client authentication

\[
\text{sign}_{sk_A}(\text{session\_hash, cert}_A, \ldots)
\]

\[
\text{session\_hash} = H(nc\ ns\ \ldots)
\]

\[
\text{session\_hash} = H(nc\ ns\ \ldots)
\]
Attacking client authentication

Alice (Client)  

Request authentication  

Client authentication  

$\text{sign}_{sk_A}(session\_hash, cert_A, \ldots)$

Bob (mybank.com)

Request authentication  

Client authentication  

$\text{sign}_{sk_A}(session\_hash, cert_A, \ldots)$

session_hash = $H(nc\ ns\ \ldots)$  

session_hash = $H(nc\ ns\ \ldots)$
Attacking client authentication

Alice
(Client)

Charlie
(evil.com)

Give Charlie all my money!

Sure thing, Alice.

Bob
(mybank.com)
Cause and mitigation

- Prime example of an attack that can arise because of the interaction of modes
- No binding between the client signature and session for which it is intended
- Complicated to find
  - requires 18 messages to set up
  - involves 2 handshakes, 2 resumptions, 1 client auth...
- Communicated this to the IETF TLS Working Group...
Dear all,

We [1] are in the process of performing an automated symbolic analysis of the TLS 1.3 specification draft (revision 10) using the Tamarin prover [2], which is a tool for automated security protocol analysis.

While revision 10 does not yet appear to permit certificate-based client authentication in PSK (and in particular resumption using PSK), we modelled what we believe is the intended functionality. By enabling client authentication either in the initial handshake, or with a post-handshake signature over the handshake hash, our Tamarin analysis finds an attack. The result is a complete breakage of client authentication, as the attacker can impersonate a client when communicating with a server:

![Diagram of TLS handshake](https://www.ietf.org/mail-archive/web/tls/current/msg18215.html)
IETF WG mailing list reactions

“Nice analysis! I think that the composition of different mechanisms in the protocol is likely to be where many subtle issues lie, and analyses like this one support that concern.”

“Thanks for posting this. It's great to see people doing real formal analysis of the TLS 1.3 draft; this is really helpful in guiding the design.”

“The result motivates and confirms the need to modify the handshake hashes to contain the server Finished when we add post-handshake authentication as is done in PR#316, which of course we'll be discussing in Yokohama.”
The Future?
What I didn’t talk about...

- In parallel, we work on computational (cryptographic) models and proofs
- More fine-grained guarantees…
  … in the property and models
- **BUT:** Manual (pen and paper) proofs are often surprisingly coarse
  - many side cases not considered well
  - ongoing work on automation, but often partial or hard to scale
- Ongoing: first cryptographic proof of the core of the Signal Protocol
  - As used by TextSecure, Facebook, WhatsApp, …
  - Claims “future secrecy”… (See also our CSF 2016 paper on Post-Compromise Security)
Take away

• People design complex systems; hard to be confident

• Formal methods tools one way of increasing confidence in solutions
  – Now at a level where we impact real-world standards
  – Careful: One methodology not enough to provide high assurance; too error-prone

• Our tools all open source (github)
  – see my webpage etc. or drop me a mail (cas.cremers@cs.ox.ac.uk)