TAMARIN

Tom Chothia

Introduction

- An introduction to the Tamarin tool
- How Tamarin works
- Sapic: applied pi-caluclus in the Tamarin tool
- Case study: testing the WPA security specification

ProVerif's Horn Clauses

- ProVerif translates applied pi-calculus to horn clauses
- These are a form of rewrite rule
- Add this to your file to see the horn clauses
  param verboseCompleted = true.
- ProVerif uses these rewrite rules to build proof trees.

Tamarin

- The tamarin tools let's you model protocol using rewrite rules.
- Each rules rewrite facts and can represents 1 step of the protocol.
- Rules take the form
- [Inputs] –[rule label] –> [Outputs]
- We can then test a wide range of security properties by testing lemmas about this rules.

Some Tamarin Facts

- In(x) : x is received as input, only used on the lefthand side.
- Out(x) : x is outputted, only used on the righthand side
- Fr(~x) : ~x is a new fresh name. Only used on lefthand side
- K(x) : The attacker knows x, only used internally.
- StateName(x) : A fact representing the current state of the protocol
- !StateName(x) : A long lived fact, that is not consumed when used. Good for, e.g., long term keys.

Tamarin Rules

You write the model directly using rewrite rules.

[Inputs] –[rule label] – [Outputs]

E.g.:

A -> B: x
B -> A: $\{x\}_{Ka}$

[In (x), !LongTermKey(A,key)]
–[A_encrypts(x)]-> [Out(senc{x}key]

Lemma

The rules fired will produce a trace of the rule labels:

```
Lablel1@t1, Label2@t2, Label3@t3,...
```

Secure properties are stated as Lemmas on these traces. e.g.,

```
lemma finishes:
exists-trace
"Ex ID_A ID_B Na Nb sec #i .
    I_finished(ID_A, ID_B, Na, Nb, sec)@i
```

Lemmas

Lemmas can express a wide range of interested security properties, e.g.,

```
lemma auth_for_I:
"All ID_A ID_B Na Nb sec #i.
  I_finished(ID_A, ID_B, Na, Nb, sec)@i
    ==> Ex #j.R_finished(ID_A,ID_B,Na,Nb, sec)@j
              & j < i"

lemma forward_secrecy:
"All ID_A ID_B Na Nb sec #j.
( R_finished(ID_A, ID_B, Na, Nb, sec)@j
   & not( Ex #l. Compromised(ID_A)@l & l <j )
   & not( Ex #k. Compromised(ID_B)@k & k <j ) )
       ==> not( Ex #i. KU(sec)@i)"
```

Restrictions

Restrictions impose conditions on the possible traces. Any

```
restriction Equality:
    "All x y #i. Eq(x,y) @#i ==> x = y"

restriction NotEqual:
    "All x y #i. Neq(x,y) @#i ==> x =/= y"

restriction unique:
    "All x #i #j. UniqueFact(x) @#i &
              UniqueFact(x) @#j ==> #i = #j"
```

Some Tamarin Tips

- "~" means fresh name only, must be used in Fr fact, can be used anywhere to restrict the name.
- To allow dishonest parties include a comprised key rules.
- Speed up your model by avoiding backtracking, i.e., make sure the protocol state fact has all the information needed to tell when a branch won't work.

Tamarin vs ProVerif IMHO

ProVerif has a nicer protocol language.

Tamarin rules are much better for complex state.

None termination in ProVerif is very hard to fix.

Tamarin's interactive model helps you understand what is going on.

Tamarin is a more developed tool.

Installing ProVerif on an ARM Mac

Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties

Benedikt Schmidt, Simon Meier, Cas Cremers, David Basin
Institute of Information Security, ETH Zurich, Switzerland

***Abstract*—We present a general approach for the symbolic analysis of security protocols that use Diffie-Hellman exponentiation to achieve advanced security properties. We model protocols as multiset rewriting systems and security properties as first-order formulas. We analyze them using a novel constraint-solving algorithm that supports both falsification and verification, even in the presence of an unbounded number of protocol sessions. The algorithm exploits the finite variant property and builds on ideas from strand spaces and proof normal forms. We demonstrate the scope and the effectiveness of our algorithm on non-trivial case studies. For example, the algorithm successfully verifies the NAXOS protocol with respect to a symbolic version of the eCK security model.**

I. Introduction

Authenticated Key Exchange (AKE) protocols are widely used components in modern network infrastructures. They assume a Public-Key Infrastructure and use the public keys to establish shared session keys over an untrusted channel. Recent AKE protocols use Diffie-Hellman (DH) exponentiation to achieve advanced security properties, namely secrecy and authentication properties in the presence of adversaries who are significantly more powerful than the classical Dolev-Yao adversary. For example, in the eCK model [1], the adversary

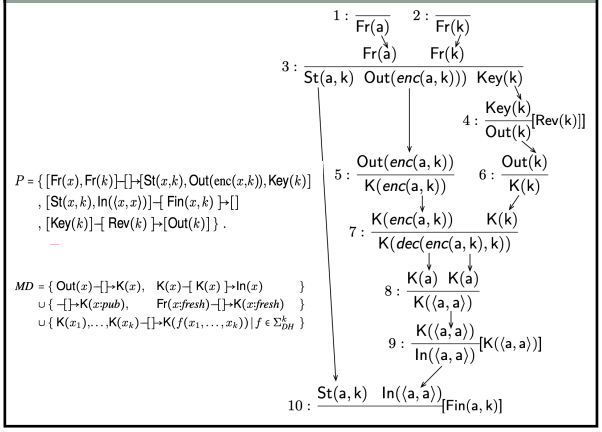
verification, which is capable of automatically verifying AKE protocols in models as described above.

Contributions: First, we give an expressive and general security protocol model, which uses multiset rewriting to specify protocols and adversary capabilities, a guarded fragment [17] of first-order logic to specify security properties, and equational theories to model the algebraic properties of cryptographic operators.

Second, we give a novel constraint-solving algorithm for the falsification and verification of security protocols specified in our model for an unbounded number of sessions. We give a full proof of its correctness along with proofs of all theorems and assertions in this paper in the extended version [18].

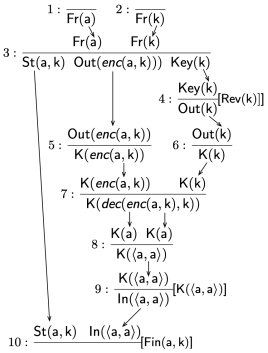
Third, we implemented our algorithm in a tool, the Tamarin prover [19], and validated its effectiveness on a number of non-trivial case studies. Despite the undecidability of the verification problem, our algorithm performs well: it terminates in the vast majority of cases, and the times for falsification and verification are in the range of a few seconds. This makes Tamarin well-suited for the automated analysis of security protocols that use DH exponentiation to achieve advanced security properties.

Organization: We introduce notation in Section II and



How to avoid loops?

- To prove you can't reach a g possible combinations of rule
- But rules such as encryption decryption can loop.
- How can we stop the proof s



Coerce rule: Coerce $\frac{\mathsf{K}^{\downarrow}_{\varepsilon}(x)}{\mathsf{K}^{\uparrow}_{\varepsilon}(x)}$ *Communication rules:* Irecv $\frac{\mathsf{Out}(x)}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x)}$ Isend $\frac{\mathsf{K}^{\downarrow}_{\varepsilon}(x)}{\mathsf{In}(x)}[\mathsf{K}(x)]$

Construction rules:

$$\frac{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(x)\ \mathsf{K}^{\uparrow}_{\varepsilon}(y)}{\mathsf{K}^{\uparrow}_{\mathsf{noexp}}(x\hat{}y)} \quad \frac{}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(x{:}pub)} \quad \frac{\mathsf{Fr}(x{:}fresh)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(x{:}fresh)} \quad \frac{\mathsf{K}^{\uparrow}_{\varepsilon}(x)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(x^{-1})} \quad \frac{}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(1)} \quad \frac{\mathsf{K}^{\uparrow}_{\varepsilon_1}(x)\ \mathsf{K}^{\uparrow}_{\varepsilon_2}(y)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathsf{enc}(x,y))} \quad \frac{\mathsf{K}^{\uparrow}_{\varepsilon_1}(x)\ \mathsf{K}^{\uparrow}_{\varepsilon_2}(y)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathsf{dec}(x,y))}$$

$$\frac{\mathsf{K}^{\uparrow}_{\varepsilon}(x)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathsf{h}(x))} \quad \frac{\mathsf{K}^{\uparrow}_{\varepsilon}(x)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathsf{fst}(x))} \quad \frac{\mathsf{K}^{\uparrow}_{\varepsilon}(x)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathsf{snd}(x))} \quad \frac{\mathsf{K}^{\uparrow}_{\varepsilon_1}(x)\ \mathsf{K}^{\uparrow}_{\varepsilon_2}(y)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}((x,y))} \quad \frac{\mathsf{K}^{\uparrow}_{\varepsilon_1}(x_1)\ \ldots\ \mathsf{K}^{\uparrow}_{\varepsilon_n}(x_n)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}((x_1 * \ldots * x_n) * (x_{n+1} * \ldots * x_l)^{-1})} \quad \frac{\mathsf{K}^{\uparrow}_{\varepsilon_{n+1}}(x_{n+1})\ \ldots\ \mathsf{K}^{\uparrow}_{\varepsilon_l}(x_l)}{}$$

Deconstruction rules:

$$\frac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x\hat{}y)\ \mathsf{K}^{\downarrow}_{\varepsilon}(y^{-1})}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x)} \quad \frac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x\hat{}y^{-1})\ \mathsf{K}^{\downarrow}_{\varepsilon}(y)}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x)} \quad \frac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x\hat{}(y*z^{-1}))\ \mathsf{K}^{\downarrow}_{\varepsilon}(y^{-1}*z)}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x)}$$

$$\frac{\mathsf{K}^{\downarrow}_{\varepsilon}((x,y))}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x)} \quad \frac{\mathsf{K}^{\downarrow}_{\varepsilon}((x,y))}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(y)} \quad \frac{\mathsf{K}^{\downarrow}_{\varepsilon}(x^{-1})}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x)} \quad \frac{\mathsf{K}^{\downarrow}_{\varepsilon_2}(\mathsf{enc}(x,y))\ \mathsf{K}^{\downarrow}_{\varepsilon_2}(y)}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x)}$$

Exponentiation rules:

$$\frac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x\hat{}y)\ \mathsf{K}^{\uparrow}_{\varepsilon}(z)}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x\hat{}(y*z))} \quad \frac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x\hat{}y)\ \mathsf{K}^{\downarrow}_{\varepsilon}(y^{-1}*z)}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x\hat{}z)} \quad \ldots \quad \frac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x\hat{}(y*z^{-1}))\ \mathsf{K}^{\downarrow}_{\varepsilon}(a*b^{-1})}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x\hat{}(y*a*(z*b)^{-1}))}$$

Case study: WPA specification

Modelling of 802.11 4-Way Handshake Attacks and Analysis of Security Properties

Rajiv Ranjan Singh[1,2](0000−0003−1868−3403), José Moreira[1](0000−0002−3210−4504), Tom Chothia[1], and Mark D. Ryan[2]

[1] School of Computer Science, University of Birmingham, Birmingham, UK
{r.r.singh, j.moreira-sanchez, t.p.chothia, m.d.ryan}@cs.bham.ac.uk
[2] Dept. of Computer Science, Shyam Lal College (Eve.), Univ. of Delhi, Delhi, India
rrsingh@shyamlale.du.ac.in

Abstract. The IEEE 802.11 standard defines a 4-way handshake between a supplicant and authenticator for secure communication. Many attacks such as KRACK, cipher downgrades, and key recovery attacks have been recently discovered against it. These attacks raise the question as to whether the implementation violates one of the required security properties or whether the security properties are insufficient. To the best of our knowledge, this is the first work that shows how to answer this question using formal methods. We model and analyse a variety of these attacks using the Tamarin prover against the security properties mandated by the standard for the 4-way handshake. This lets us see which security properties are violated. We find that our Tamarin models vulnerable to the KRACK attacks do not violate any of the standard's security properties, indicating that the properties, as specified by the standard, are insufficient. We propose an additional security property and show that it is violated by systems vulnerable to KRACK attacks, and that enforcing this property is successful in stopping them. We demonstrate how to use Tamarin to automatically test the adequacy of a set of security properties against attacks, and that the suggested mitigations make 802.11 secure against these attacks.

Keywords: IEEE 802.11 · WPA2 · 4-way handshake · Group key handshake · KRACK attack · Downgrade attack · Tamarin prover · SAPiC.

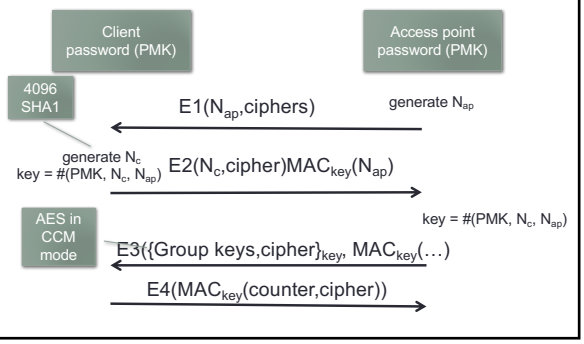
1 Introduction

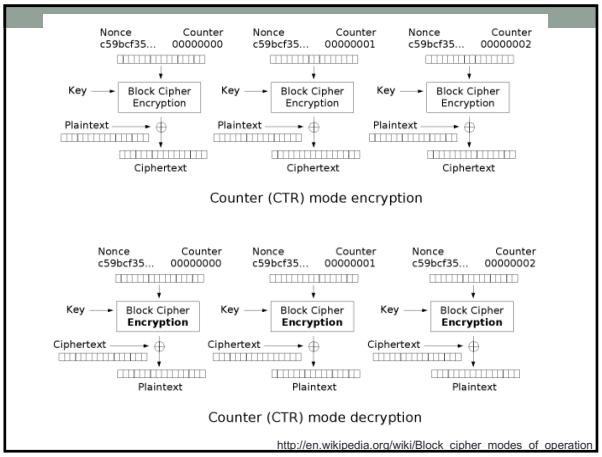
The IEEE 802.11 standard [3] defines a *4-way handshake* as the key management

WPA2 wi-fi security

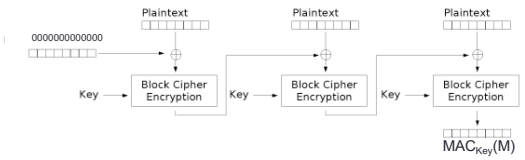
- Wi-fi security protocols developed by a closed, paid membership group of companies.
- First wi-fi protocol: WEP completely broken.
- Second wi-fi protocol: WPA used own cipher, broken
- Third protocol WPA2 uses AES in CCM mode.
  - Security is based on a password known to the access point and client.
  - Password can be bruteforced offline.
  - KRACK attack
- WPA3 new proposed protocol with ECDH.

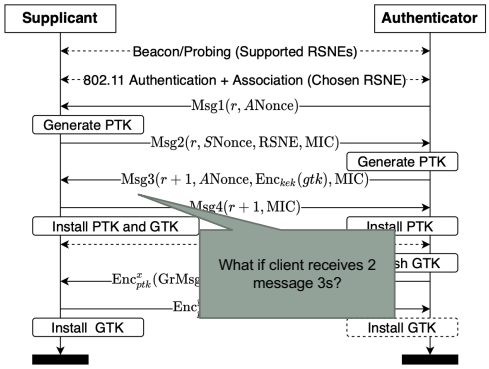
Simplified WPA 2



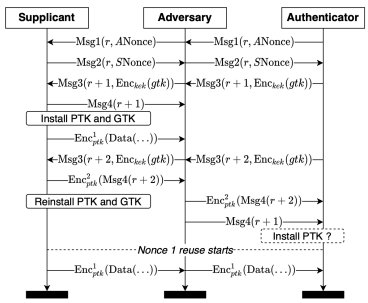


CBC MAC

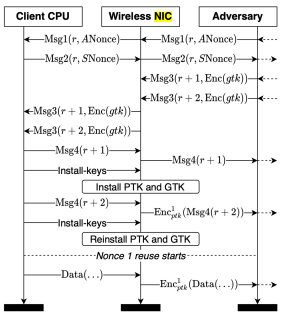




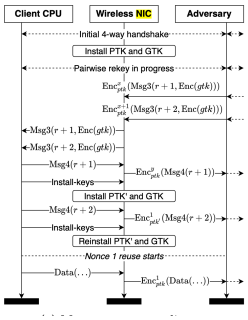
KRACK Attack Version 1



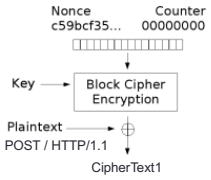
KRACK Attack Version 2



KRACK Attack Version 3



KRACK attack on WPA2

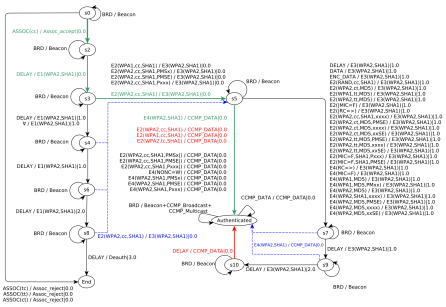


"POST / HTTP/1.1" xor E(c59 … 000) = ciphertext1

"Password: ??????" xor E(c59 … 000) = ciphertext2

Ciphertext2 xor ("POST / HTTP/1.1" xor ciphertext1)
= Ciphertext2 xor E(c59 … 000)
= "Password: ??????" xor E(c59 … 000) xor E(c59 … 000)
= "Password: ??????"

Downgrade attack



WPA Security Spec

- WPA has a very long and detailed security specification
- So what when wrong?
- We can use Tamarin's restrictions to see if these errors are due to problems in the spec or the implementation.
  - We make a model that includes these attacks.
  - We then impose the security specifications as restrictions.
  - If these restrictions stop the attack then it's an implementation problem.
  - If the restrictions don't stop the attack then it's a problem in the spec.

Security Spec as restrictions

- Confirm the existence of the PMK at the peer.

$$\forall id_1, id_2, pmk_1, pmk_2, t_1, t_2.\ \text{A\_HasPmk}(id_1, pmk_1)@t_1 \wedge \text{S\_HasPmk}(id_2, pmk_2)@t_2 \Rightarrow (pmk_1 = pmk_2). \quad \text{(ConfPmk)}$$

- Ensure that the security association keys (PTK/GTK) are fresh.

$$\forall id_1, id_2, ptk, t_1, t_2.\ \text{S\_ComputesPtk}(id_1, ptk)@t_1 \wedge \text{S\_ComputesPtk}(id_2, ptk)@t_2 \Rightarrow (tid_1 = tid_2). \quad \text{(FreshPtk)}$$

- Transfer the GTK from the Authenticator to the Supplicant.

$$\forall id, gtk, t_1.\ \text{S\_InstallsGtk}(id, gtk)@t_1 \Rightarrow (\exists t_2.\ \text{A\_GeneratesGtk}(gtk)@t_2 \wedge (t_2 < t_1)). \quad \text{(SameGtk)}$$

Security Spec as restrictions

- Synchronise the installation of temporal keys

$$\forall S_{\text{id}}, A_{\text{id}}, pars, t_1.\ \text{S\_CommitPtk}(S_{\text{id}}, A_{\text{id}}, pars)@t_1 \Rightarrow ((\exists t_2.\ \text{A\_RunningPtk}(A_{\text{id}}, S_{\text{id}}, pars)@t_2 \wedge (t_2 < t_1)) \wedge \neg(\exists S'_{\text{id}}, A'_{\text{id}}, t_3.\ \text{S\_CommitPtk}(S'_{\text{id}}, A'_{\text{id}}, pars)@t_3 \wedge \neg(t_3 = t_1))). \quad \text{(AgreePtk)}$$

$$\forall S_{\text{id}}, A_{\text{id}}, pars, t_1.\ \text{S\_WCommitGtk}(S_{\text{id}}, A_{\text{id}}, pars)@t_1 \Rightarrow ((\exists t_2.\ \text{A\_WRunningGtk}(A_{\text{id}}, S_{\text{id}}, pars)@t_2 \wedge (t_2 < t_1)), \quad \text{(WeakAgreeGtk)}$$

$$\forall id, ptk, t_1.\ \text{S\_InstallsPtk}(id, ptk)@t_1 \Rightarrow \neg(\exists t_2.\ K(ptk)@t_2). \quad \text{(SecretPtk)}$$

- Confirm the selection of cipher suites.
  - As AgreePtk

SAPIC

$\langle P,Q \rangle$ ::=	processes
$\mid 0$	terminal (null) process
$\mid P \mid Q$	parallel composition of P and Q
$\mid\ !P$	replication of P
$\mid\ \nu\ a;\ P$	binds a to a new fresh value in P
$\mid\ \mathsf{out}(m,t);\ P$	outputs message t on channel m
$\mid\ \mathsf{in}(m,t);\ P$	inputs of message t on channel m
$\mid$ if $Pred$ then P [else Q]	P if predicate $Pred$ holds; otherwise Q
$\mid$ event $F;\ P$	executes event (action fact) F
$\mid\ P + Q$	non-deterministic choice
$\mid$ insert $m,\ t;\ P$	inserts t at memory cell m
$\mid$ delete $m;\ P$	deletes the content m
$\mid$ lookup m as x in P [else Q]	if m exists, bind it to x in P; otw. Q
$\mid$ lock $m;\ P$	gain exclusive access to cell m
$\mid$ unlock $m;\ P$	waive exclusive access to m
$\mid\ [L] -[A]\!\rightarrow [R];\ P \quad (L,R,A \in \mathcal{F}^*)$	provides access to TAMARIN MSRs

Testing the security spec.

Security Property	a) ConfPmk	b) FreshKeys		c) SynchronisedKeys				d) SameGTK	e) ConfCiphers
Lemmas	(ConfPmk)	(FreshPtk)	(FreshGtk)	(AgreePtk)	(AgreeGtk)	(WeakAgreeGtk)	(SecretPtk)	(SecretGtk)	(SameGtk)	(AgreeCs)
PTK reinst. Figs. 4, 5	✓	✓	✓	✓	✓	–	✓	✓	✓	✓
PTK reinst. Fig. 6	✓	✓	✓	✓	✓	–	✓	✓	✓	✓
PTK reinst. Fig. 7	✓	✓	✓	✓	✓	–	✓	✓	✓	✓
GTK reinst. Fig. 8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Downgrade Fig. 9	✓	✓	✓	✓	✓	–	✓	✓	✓	✗

✓: attack still exists ✗: attack is stopped

Conclusion

- Automatic checking with ProVerif and Tamarin make checking protocol much easier.
- Also much less error prone
- ProVerif uses the applied pi-calculus
- Tamarin is a more advanced tool with more features.

Homework

- Use Tamarin or ProVerif to model and check your protocol.
- A very good idea to look at and copy parts of the example files.
- You will need to decide what the security properties should be.
  - Always include sanity check events/lemmas in make sure your model terminates.
  - If it doesn't more the events back, until until you find the error.