Master Thesis Tamarin Notebook
Information Security Group - Prof. D. Basin

Jonas Passweg

May 2023

1 Overview

Research group: Formal Verification with the Tamarin Prover (Student projects / Research website)
Tamarin prover: security protocol verification tool (Repo and Publications)
Master thesis: Unification modulo homomorphic encryption in the Tamarin Prover
Supervisors: Sofia Giampietro, Dr. Ralf Sasse, Prof. Dr. David Basin
Task: Extend Tamarin to support homomorphic encryption with an existing unification algorithm

2 Theory and Notes

2.1 Basics of Term Rewriting

Source: Formal Methods for Information Security
- Literature: TODO go over literature on the website.
- Lecture 5: Protocol Basics, Adversary Model, Different Attacks
- Lecture 6: Term Rewriting
- Lecture 7: Protocol Syntax, Semantics, Properties (Secrecy not summarized)
- Lecture 8: Continued, Tamarin Overview (Authentication not summarized)
- Lecture 9: Algorithmic verification (Undecidability in general sense not summarized)

General math:
- multiset $[0 \to 2] = [0, 0]$
- $op^{\#}$ is the same operator as with sets, but for multisets

Algebra:
- Signature Σ = set of function symbols with arity $n \geq 0$
- Algebra $T_\Sigma(X, N)$ over Σ with X = variables and N = names
- Set of ground terms T_Σ consists of terms build without variables, i.e., $T_\Sigma := T_\Sigma(\emptyset, N)$
- Equation $t = t'$, oriented $t \to t' \in \overrightarrow{E}$ or $t \leftarrow t' \in \overleftarrow{E}$
- Equational Theory (Σ, E)
- Free Algebra $t_1 =_{free} t_2$ iff $t_1 =_{syntactic} t_2$

Algebra properties:
- Congruence relation $=_E$
- Equivalence class $[t]_E$ of term t modulo E: All terms equal to t according to congruence relation $=_E$
- Quotient algebra $T_\Sigma(X, N)/_{=_E}$ interprets each term by its equivalence class
- Confluent = if you rewrite t as s1 and separately as s2, there exists a t' such that one can rewrite both s1 and s2 to t'. [6]
- Terminating = no infinite sequence rewrite rules to rewrite a term
- Convergent = terminating and confluent
- Normal form = if an equational theory is convergent, every term has a unique normal form $t\downarrow$
- If $(\Sigma, \overrightarrow{E})$ then $t =_E u$ iff $t\downarrow = u\downarrow$
- Coherent Modulo = doesn't lose equivalence after rewriting [9]

Cryptographic Messages:
- X = set of variables $A, B, X, Y, Z, ...$
- PV = set of public values $a, b, c, ...(agents, ...) \subseteq N$
- FV = set of fresh values $na, nb, l, ...(nonceskeys, ...) \subseteq N$
- $\{\}$ = asymmetric en/decryption, $\{||\}$ = symmetric en/decryption

Substitution and Unification:
- Substitution $\sigma : X \to T_\Sigma(X, N)$, in postfix notation

- Composition $\sigma\tau$ = apply σ then τ
- Term position $t|_{[a_1,...,a_N]}$ is the subterm at position $[a_1, ..., a_N]$
- Matching t matches I if there is a substitution σ so that $t = I\sigma$
- Term replacement $t[u]_p$ is the term t by replacing $t|_p$ with u.
- Rewriting $I \to r$ is a rewrite rule applicable to t if I matches a subterm $t|_p$ with $t|_p = I\sigma$. The resulting rewrite step is $t \to t[r\sigma]_p$
- t and t' unifiable in (Σ, E) if there is substitution σ (=unifier) such that $t\sigma =_E t'\sigma$
- E-equality step $u \to_{\overrightarrow{E} \cup \overleftarrow{E}} v$ denoted as $u \leftrightarrow_E v$
- Equality proof $t_0 \leftrightarrow_E t_1 ... \leftrightarrow_E t_n = t_0 \leftrightarrow_E^* t_n$

Multiset Rewriting:

- Fact Symbols Σ_{fact}
- Fact $F(t_1, ..., t_k)$ with $F \in \Sigma_{fact}$ and $t_1, ..., t_l \in T_\Sigma(X, N)$
- lin(l) denotes the multiset of linear facts in l and per(l) denotes the set of persistent facts in l (often denoted by an ! in front of the fact).
- Labeled multiset rewriting rule (MSR) $l \xrightarrow{a} r$ with l, r, a are multiset of facts
- A labeled multiset rewrite system is a set of MSRs
- Instance of an object X is the result of applying a substitution σ to all terms in X, written $X\sigma$
- Ground instance of X is an instance where all terms are ground (i.e. do not include variables)
- $ginsts(R)$ is the set of all ground instances of rules in the multiset rewrite system R
- $\mathcal{G}$ is the set of ground facts. $\mathcal{G}^\#$ is the set of finite multisets of elements from $\mathcal{G}$
- A state is a finite multiset of ground facts (i.e. an element of $\mathcal{G}^\#$). Eg: $[St\_R\_1(A, 17, k_1), Out(K_1), K(m)]$
- For a MSR system R we define the labeled transition relation $steps(R) \subseteq \mathcal{G}^\# \times ginsts(R) \times \mathcal{G}^\#$ with for $(S, l \xrightarrow{a} r, S') \in steps(R)$: $lin(l) \subseteq^\# S$, $per(l) \subseteq S$, and $S' = (S \setminus^\# lin(l)) \cup^\# r$
- An execution of R is an alternating sequence $S_0, (l_1 \xrightarrow{a_1} r_1), S_1, ...S_k$ with $S_0 = \emptyset^\#$, $(S_{i-1}, l_i \xrightarrow{a_i} r_i, S_i) \in steps(R)$, and $(l_i \xrightarrow{a_i} r_i) = (l_j \xrightarrow{a_j} r_j) = ([] \to [Fr(n)]) \Rightarrow i = j$
- A trace for an execution is $a_1, ..., a_k$

Specific multiset rewriting rules:

- Adversary rules: $\frac{Out(x)}{K(x)}$, $\frac{K(x)}{In(x)}K(x)$, $\frac{Fr(x)}{K(x)}$, $\frac{K(t_1)...K(t_k)}{K(f(t_1,...,t_k))}$
- Fresh rule: $[] \to [Fr(N)]$, with N unique
- State agent fact: $St\_R\_s(A, id, k_1, ..., k_n)$ with R = role, s = protocol step, A = name of agent, id = thread identifier, k_i are terms kept in agent's storage
- Protocol rule: l = In, Fr and agent state facts, r = Out and agent state facts. Either In or Out, but not both in a rule. Agent state facts increment in rule. Every variable in r must occur in l.
- Infrastructure rule: rules like public key generation

Protocol properties:

- Semantics of a security protocol P is a set of traces $||P|| = traces(P)$
- Protocol P satisfies property ϕ, written $P \models \phi$, iff $||P|| \subseteq ||\phi||$
- Attack traces are those in $||P|| - ||\phi||$
- Claim events $Claim\_claimtype(A, t)$. Eg: $Claim\_secret(A, N_A)$ claims that N_A is a secret for agent A
- timestamped event $F@i$ holds on traces $tr = a_1, ..., a_n$ if $F \in a_i$
- Constraint system Γ is a set of formula, node and/or edge constraints.
- A solution to a constraint system Γ is a pair $(dg, \theta) \models \Gamma$, with dg being a dependency graph and θ being a valuation, assigning dg's indices to index variables and ground terms to term variables.
- The set of solutions is denoted as $sol(\Gamma)$.

Cryptography:

- cryptographic primitives = low-level cryptographic algorithm. Symmetric encryption, hashing, signing, and public key encryption
- symbolic model = cryptographic primitives are represented by function symbols considered as black boxes, the messages are terms on these primitives, and the adversary is restricted to compute only using these primitives [3].

2.2 Tamarin Theory

Sources:
Formal Methods for Information Security
Paper: Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties
Overview:

- Protocol: Uses multiset rewriting to represent protocol. Adversary message deduction rules given as multiset rewriting rules
- Properties specified in first-order logic (exists, for all)

- Backwards reachability analysis – searching for insecure states (negate security properts, search for solution)
- Constraint solving (uses Normal dependency graphs)
- Internal workflow: See image in attachments

Automated Analysis of Diffie-Hellman
Protocols and Advanced Security Properties: Introduction and Notations:
Skipped

2.3 Haskell

2.4 Tamarin Code

REAME.md: describes the organization of the repository of the Tamarin prover
CONTRIBUTING.md: Information about contributing
Haskell Code style: link
Code src folder:

- Main.hs: main file
- src/Main/TheoryLoader.hs: intruder rules for Diffie-Hellman exponentiation and Bilinear-Pairing

Code Lib folder

- TODO

Misc Files and Folders Info:

- tamarin-prover.cabal: Haskell package info file
- /etc/: spthy syntax highlighting
- /examples/: example protocol models
- regressionTests.py: tests protocols in case-studies-regression folder
- /case-studies-regression/: filled with analyzed protocols from examples folder
- /doc/: some dev notes and notes on regressionTests

2.5 Unification Algorithms

- unification algorithm = an algorithm for determining the substitutions needed to make two predicate calculus expressions match by using substitution [13, 12].
- folding variant narrowing [5]
- finite variant property (FVP) = (needed for folding variant narrowing) [4]

2.6 Finite Variant Property

[4] TODO: check why important

2.7 Folding Variant Narrowing

Narrowing = If a set of equations EuAx is such that E is confluent, terminating, and coherent modulo Ax, narrowing with E modulo Ax provides a complete EuAx-unification algorithm. [7, 5]
TODO: continue with the introduction

2.8 Cap unification

Paper [1]:

ABSTRACT:
We address the insecurity problem for cryptographic protocols, for an active intruder and a bounded number of sessions.

- bounded number of sessions. Don't we need to prove it for unbounded number of sessions?
- with [11] several sessions can be reduced to one: the paper only looks at 1 session, i.e., every protocol rule is only used once.
- uses this to model every step of a protocol session as a Cap Constraint

2. SETTING THE STAGE
The homomorphic encryption theory that we consider – denoted as HE in the sequel – extends DY with the following rule:

- do we also need to extend intruder

constructor system = constructors + defined symbols
HE Active and passive deductions:

- $\pi_1(p(x,y)) \to x$
- $\pi_2(p(x,y)) \to y$
- $+$
- $e(p(x,y),z) \to p(e(x,z),e(y,z))$ (E_h)

Mistake at start of page 4: The strand trace for S is ..., the first term should be $-(p(p(A,B),N_a)$

E_h-Unifn: shaping and parsing rules, failure rules, plus rules for unification Unification rules on end page 7 and start of page 8
HE+:

- HE
- $+$
- $d(P_v(e(x_1,z),...,e(x_n,z)),z) \to P_v(x_1,...,x_n)$

Conclusion: "reduce" deduction modulo HE+ to E_h for which unification is essentially snytactic

- Where is that reduction written down?
- Does that help us since Tamarin tries to be very close to syntactic unification with finitely many variants.

Cap Unification Algorithm

```
4. Unification Modulo EpsilonH

Precondifiton:
Equations are given in EpsilonH normal form
Unification(Γ)

Rules for Rules:
Always apply Failure first
Always apply Shaping/Failure/Parsing/Variable Substitution before any other rules

Rules:
Trivial             t=t and Γ                          => Γ
Std Decomposition   f(s1,...,sm)=f(t1,...,tm) and Γ => Γ and {s1=t1} ... {sm=tm}
Variable Substitution
                    x=t and Γ, x notin Vars(t)      => Γphi and {x=t}
                    phi = [x -> t]
Clash               f(s1,...,sm)=g(t1,...,tn) and Γ => fail
                    f!=g and not (f=p|e and g=e|p)
Occur Check         x=t and Γ, t!=x and x in Vars(t)=> fail
Shaping             Pv(t1,...,E(x,km,...,kn),...tl) = E(s,k1',...,kn') and Γ
                    => Γ and Pv(t1...,E(x',k1',...,km-1',km,...,kn),...,tl)
                        = E(s,k1',...,kn')
                    and x = E(x',k1',...,k(m-1)')
                    where x' is a fresh variable, v is legal bit string and n >= m > 1
Failure1            s=t and Γ                          => fail
                    s,t are out of phase on some variable at a non-key position
Failure2            Pv(t1,...,E(t1,k1,...,km),...,tk) = E(s,k1',...,kn') and Γ
                                                       => fail
                    where v is legal bit string, ti not a variable and m<n
Parsing
        Pv(E(t1,k11,...,k1(m1)),...,E(tl,kl1,...,kl(ml))) = E(s,k1,...,km) and Γ
        => Γ
        and Pv(E(t1,k11,...,k1(m1-1)),...,E(tl,kl1,...,kl(ml-1)) = E(s,k1,...,k(m-1))
        and k1(m1) = k2(m2) = ... = kl(ml) = km
        where v is legal bit string

Helper functions:

solved(Γ): Γ is in solved form: Γ is empty of of the form {x1=t1,...,xn=tn},
for which each xi only appears on the left side of the equation, and ti is a usual
term

Vars(t): variables in t
```

```
pos(t):
        t is a variable = {e}
        t is a function f(t1,...,tn) = {e} and {ip | p in pos(ti)}

ppos(q,t):
        (e,t) = e
        (iq,t) = i ppos(q,t|i)  if the symbol at the top of t is p
        (iq,t) = ppos(q,t|i) if the symbol at the top of t is not p


epos(q,t):
        (e,t) = e
        (iq,t) = i epos(q,t|i)  if the symbol at the top of t is e
        (iq,t) = epos(q,t|i) if the symbol at the top of t is not e

incompatible(q1,t1,q2,t2):
        if ppos(q1,t1) is a proper prefix of ppos(q2,t2)
        or  (ppos(q1,t1) = ppos(q2,t2), epos(q1,t1) != epos(q2,t2) and
            (epos(q1,t1) and epos(q2,t2) only contain 1's in their sequences)

in_phase(t1,t2):
        for all positions q1 in t1 and q2 in t2 such that t1|q1 = t2|q2:
            not incompatible(q1,t1,q2,t2)

out_of_phase(t1,t2,x):
        exists q1,q2: t1|q1 = t2|q2 = x and incompatible(q1,t1,q2,t2)

non_key(q,t):
        epos(q,t) contains only 1's

valid_bit_string(v):
        if v is empty string
        or v = 1a1,...,1an,2b1,...,2bm with a1,...,an and b1,...,bm both valid bit strings

pure_p_position(q,t):
        if epos(q,t) = e

maximal_pure_p_position(q,t):
        if epos(q,t) = e and q is not a proper prefix of a position for which epos(q,t) = e

P(t):
        {q1,...,qn} of lexicographically ordered maximal_pure_p_position in t
        and (t1,...,tn) for ti = t|qi

pure_e_position(q,t):
        if ppos(q,t) = e

penuk_position(q,t):
        if ppos(q,t) = e and q contains no 2 at all or contains 2 only as the last element

maximal_penuk_position(q,t):
        if penuk_position(q,t) and q is not a proper prefix of any penuk-position in t

E(t):
        {q1,...,qn} of lexicographically ordered maximal_penuk_position  in t
        and (t1,...,tn) for ti = t|qi
```

3 Meetings

3.1 Monday 08.05.2023 16:00 - Introductory meeting

Notes before the meeting:

- I found out I do not have enough knowledge about the terminology used in the symbolic analysis.
- I am still interested in having 3-4 weeks to prepare before the thesis starts.
- I focused on understanding papers but especially formal methods for information security course

Questions:

- Do you recommend any reading or preparation like [10, 2]
- For the time plan after 2 weeks, what are the expected difficulties

Answers:

- I'm going to receive recommendations per email:
  Term rewriting: Section 2.2,2.3 (page 35-39) of Schmidt's thesis gives a nice quick overview [10].
  Tamarin: CSF paper in which Tamarin was introduced by Schmidt and all [11]. More details are in Section 3 of Schmidt's thesis. [10]
- most difficult part will be combining unification algorithm modulo homomorphic encryption rules with tamarin

Notes:

- start date set to 29.05.2023
- for theories that satisfy the FVP, Tamarin can create all (a finite number of) variants of a term (by calling MAUDE-NPA) and then compare the output variants of each rule with the attacker's goal using equational theory modulo nothing. However, the homomorphic encryption theory does not satisfy FVP. Therefore, the goal is to compare every variant of the rewrite rules with equational theory modulo the homomorphic encryption.

Tasks:

- Finish reading course material Formal Methods for Information Security (DONE)
- Read recommended readings from Sofia (got an overview)
- find out what you need to understand and how to summarize the information (Started)
- Maybe install Haskell and try it out, and implement a straightforward unification algorithm
- Switch links in the document with BibTeX (DONE)
- Also look at this other thesis: [8]

3.2 Wednesday 24.05.2023 14:00

What I did:

- Read the recommended readings
- Upon realizing the lack of basic knowledge, I went through more of the course slides (Formal Methods for Information Security)
- Paper understood everything up until page 7. On my third read-through

Ideas:

- Figuring out why we can't have for the homomorphic encryption like with the adversary different rules (up-arrow, down-arrow) such that it terminates.
- Connection between has to be done by the constraint rewriting algorithm and the unification algorithm. Did I understand correctly that Tamarin does not implement unification at all, since it calls Maude-NPA for the finite variants?

Questions or unclear Concepts:

- Instantiation and sort are not completely clear. Sort from: "A. Protocol Specification and Execution To model cryptographic messages, we use an order-sorted term algebra with the sort msg and two incomparable subsorts fresh and pub for fresh and public names. "
  Idea: has to do with instantiation
- AC from paper (in AC dependency graph) not looked at in depth.
- The exact definition of modulo algebra is not found. Understood, that it means that the algebra defines what is equal.

Discussed during the meeting:

- Think of sorts as types.
- Maybe don't go too much into detail in the paper. Especially concerning the cap unification algorithm paper.
- Looked at the overview graph of Tamarin. It makes sense to already take a look at the Tamarin code.

Tasks:

- Add constraint solver algorithm to this document / summarize what you read on tamarin verification from the lecture / tamarin paper
- Create time plan for master thesis
- Continue with reading about tamarin and cap unification
- Look at Tamarin code
- Tamarin has other things on their GitHub repo, like teaching. Might also take a look at that.
- automatic code overview with visual studio (needs enterprise version)

3.3 Wednesday 07.06.2023 15:00

What I did: got an overview of the codebase and created a rough graph
Goal for next week: understand which part of the code I need to change/adapt.

Code Questions:
- What does accountability mean? What are accountability lemmas in an open theory?
- What is the difference between a diff theory and a theory? Is diff == diffie hellman?
- What is an open theory? ("Open a theory by dropping the closed world assumption and values whose soundness depends on it.")
- What is exaclty part of a signature and what not?
- What is the sorryProver?
- What does in addParamsOptions the words "[Add parameters in the OpenTheory, here] openchain and saturation [in the options]" mean?
- What does "prove with version" (in TheoryLoader closeTheory) mean?

Todo Next:
- Find classes that need to be expanded and better understood for your thesis
- Might be interesting to deepen hierarchy structure of Sapic
- Put everything in scheme of the paper. Read the paper and where does what happen (eg. unification)
- Compare with dependency graph we got. very confusing
- Find out which code parts I need to change
- color code the graph for better overview

Questions concerning the tamarin prover code:
- Are all Lemmas that need to be proven accountability lemmas as mentioned in the accountability library? Why does it need a separate library?
- What does the term Sapic mean? The translate function in the Sapic Module mentions "Translates the process (singular) into a set of rules and adds them to the theory" and is used in the loadTheory function of the TheoryLoader Module after parsing and before returning the Open Theory. How can this be understood? Is it even the default theory option? The module is defined in the module Theory.Module. I was unsure, and now I'm thinking it is not. However, in the Theoryloader module there is a comment saying, "– MSR is default module, i.e., we translate by default ... otherwise we get warnings for actions used in lemmas that appear only in processes.", at least concerning the output module.
- What is a diff theory? (got answered). Do we need to consider this for our unification algorithm?
- Did I understand it correctly that an open theory is one that is not yet proven, while a closed one is proven?
- During loadTheory, addParamsOptions is called, which states that it adds the parameters open chain limits and saturation limits. Can these be ignored for the unification algorithm? This question can be ignored as I am pretty sure they are not in our focus.

Code Questions
- What does accountability mean? What are accountability lemmas in an open theory?
- What is difference between a diff theory and a theory? Is diff == diffie hellman?
- What is an open theory? ("Open a theory by dropping the closed world assumption and values whose soundness depends on it.")
- What is exaclty part of a signature and what not?
- What is the sorryProver?
- What does in addParamsOptions the words "[Add parameters in the OpenTheory, here] openchain and saturation [in the options]" mean?
- What does "prove with version" (in TheoryLoader closeTheory) mean?

Todo Next:
- Find classes that need to be expanded and better understood for your thesis
- Might be interesting to deepen hierarchy structure of Sapic
- Put everything in scheme of the paper. Read the paper and where does what happen (eg. unification)
- Compare with dependency graph we got. very confusing

- Find out which code parts I need to change
- color code the graph for better overview
- look at cap unification and write up pseudo code

Answers Sofia:

- Diff theory: able to add term subsitution and looks if both protocol traces are the same with and without subsitution etc.
- SorryProver: placeholder for prover if not yet proved
- Signatures: builtin functions and what they allow to be equal (like d(e(x,z),z) = z)
- Accountability Lemmas. Turns out this is a completely orthogonal (and quite un-used) Tamarin feature. These lemmas integrate a framework that deals with accountability (i.e. being able to blame some party for violations of given security property) in an automatic way in Tamarin. As far as I understand an accountability lemma automatically just generates multiple normal protocol lemmas that will be proven/disproven and depending on the outcome a certain party can be blamed. As far as I know (and most group members here would agree), this is a very un-used feature and we can completely ignore it for the project. I personally had completely forgotten about these. So to answer your concrete questions: these are not the user-defined lemmas that we want to prove and they are in a separate library because, as mentioned, it's an extra feature.
- Sapic. SAPIC stands for "Stateful Applied PI-Calculus". It deals with allowing to model protocols using process calculus. In particular the Sapic library translates process calculus into standard Tamarin multiset-rewrite rules. I imagine there is a flag somewhere to indicate if the protocol is being specified via Processes or Multiset-rewrite rules. As for your concrete questions: How to understand the sentence you mention is as follows: once the model given in process calculus has been parsed, it is translated into a multiset-rewrite rule (MSR) format and from there then the open theory is constructed. I think what they mean when they say that the translation is "default" is that if the input is aprocess model, by default we translate it to a MSR theory. I am not super sure what the comment you mentioned refers to, but I will look into it for Wednesday. Since most Tamarin users model protocols using multiset re-write rules, we can ignore the translation from processes for now, so I wouldn't spend too much time trying to go through this SAPIC folder (it's just a Tamarin extension to allow users to specify their protocols using processes instead of MSR rules.
- Open theories. Yes, you are correct.
- addParamsOptions. Yes, I think these can be ignored for now. We will probably not need to change this for the project.

3.4 Wednesday 14.06.2023 15:00

TODO General

- put other papers from pictures also on zf
- maudenpa alluege algorithmus
- add question what is signature
- put tamarin.txt on overleaf

TODO CODE DIAGRAM

- opentheory DONE
- put theory at good place DONE
- prover calls DONE
- proof classes DONE
- do Reduction-¿runReduction DONESOMEWHAT???
- do Sources-¿matchToGoal subcalls DONE
- do Unification function solveMatchLNTerm DONESOMEWHAT???
- term library
- do other unification functions
- find out where rules for unification are
- cut from proof class
- all items classes
- All theory.model classes
- terms library classes

Outside unification method calls

- unification outside used functions
- unifyLNTerm, unifiableLNTerms
- unifyLNTermFactored
- matchLVar in Constraint.Solver.Sources but can be ignored
- solveMatchLTerm

- solveMatchLNTerm - the one we got to on our own
- + a lot of Maude stuff

Meeting
- ask how Reduction runReduction works
- mention pretty got ignored
- highlight what is important
- tell me if I did something wrong

Goals for next week
- finish diagramm
- implement unification algorithm in tamarin

3.5 Wednesday 21.06.2023 15:00

new repo

Compile and run
- go to tamarin-prover folder
- call make default
- then call /home/jpassweg/.local/bin/tamarin-prover test

Todo
- look if verbose mode
- print what the unification algorithm returns for the test cases and what happens after avoidfresh
- compile and then recompile with a print statement to test if you compiled correctly
- alles uf overleaf
- Write test infrastructure first on your fork so that you know what your method needs to solve
- ufde tamarin-prover.github.io website hets sehr vieli interessanti papers und thesises
- find out about terms
- fork project on gitlab
- Find out which testsfile test unification and how to call test mode in tamarin
- maybe use pretty printing classes for test printing
- find out how to compile from source
- find out how to run haskell in general and do some own small tests
- make and explain graph more
- what are l and ln terms
- write down algorithm in unification file
- write down tests in separate test file or in the same test file as other unification tests
- is having v being a legal bit string needed for our unification
- find out how to integrate unify into the whole process
- I don't understand why in my version after flattenUnif it needs a $ and not a <$> like in the other unification methods

Other cap unification papers:

References

[1] Siva Anantharaman, Hai Lin, Christopher Lynch, Paliath Narendran, and Michael Rusinowitch. Cap unification: Application to protocol security modulo homomorphic encryption. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, page 192–203, New York, NY, USA, 2010. Association for Computing Machinery.

[2] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[3] Bruno Blanchet. Security protocol verification: Symbolic and computational models. In Pierpaolo Degano and Joshua D. Guttman, editors, *Principles of Security and Trust*, pages 3–29, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[4] Santiago Escobar, José Meseguer, and Ralf Sasse. Effectively checking the finite variant property. In Andrei Voronkov, editor, *Rewriting Techniques and Applications*, pages 79–93, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[5] Santiago Escobar, Ralf Sasse, and José Meseguer. Folding variant narrowing and optimal variant termination. In Peter Csaba Ölveczky, editor, *Rewriting Logic and Its Applications*, pages 52–68, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[6] Sabina Fischlin. Formalising zero-knowledge proofs in the symbolic model. Master Thesis, 9 2021.

[7] Christopher Lynch. Basic narrowing. Slides, 7 2019.

[8] Simon Meier. *Advancing automated security protocol verification.* Doctoral thesis, ETH Zurich, Zürich, 2013.

[9] Ralf Sasse, Santiago Escobar, Catherine Meadows, and José Meseguer. Protocol analysis modulo combination of theories: A case study in maude-npa. In Jorge Cuellar, Javier Lopez, Gilles Barthe, and Alexander Pretschner, editors, *Security and Trust Management*, pages 163–178, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[10] Benedikt Schmidt. *Formal analysis of key exchange protocols and physical protocols.* Doctoral thesis, ETH Zurich, Zürich, 2012.

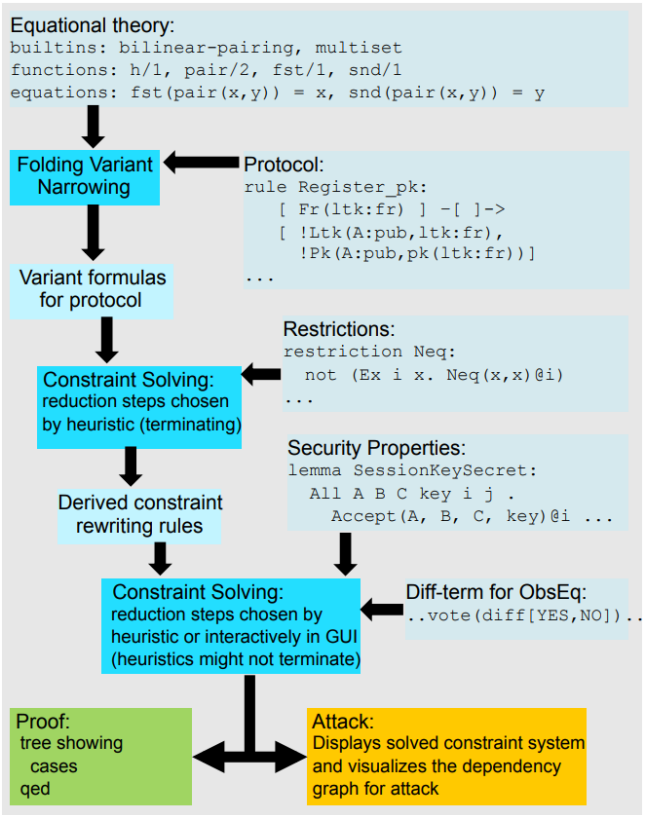
[11] Benedikt Schmidt, Simon Meier, Cas J.F. Cremers, and David Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In *IEEE 25th Computer Security Foundations Symposium (CSF 2012)*, pages 78 – 94, New York, NY, 2012. IEEE. 25th Computer Security Foundations Symposium (CSF 2012); Conference Location: Cambridge, MA, USA; Conference Date: June 25-27, 2012.

[12] Johannah Sprinz. A brief introduction to first-order logic unification at the example of corbin and bidoit's variation of robinson's unification algorithm. Munich, LMU Munich, Master Seminar "Unification" SS21, Sep 2021. `https://www.cip.ifi.lmu.de/~sprinz/sprinz_2021_unification.pdf`, Accessed: October 29, 2022.

[13] unknown. Slides on unification from lecture. C93734F252A1F5B0294E69D3A9A7AB8F.

4 Attachments

- Internal workflow of Tamarin
- Master thesis proposition
- Master thesis provisional time schedule
- Tamarin code dependencies



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Master's Thesis

Unification modulo homomorphic encryption in the Tamarin Prover

Supervisor: Sofia Giampietro, Dr. Ralf Sasse
Professor: Prof. D. Basin
Issue Date: January 2023
Submission Date: TBD

1 Introduction and Motivation

Tools for the automated analysis of security protocols, such as Tamarin [11] and ProVerif [5], have found considerable success over the past five years with the discovery of new attacks on protocols such as TLS 1.3 [8], Noise [10] and 5G [4]. However, whilst manual protocol analysis is typically carried out in the *computational model* of cryptographic primitives, automated analysis is carried out in the *symbolic model*.

The symbolic model makes much stronger assumptions about the behavior of cryptographic primitives than the computational model. Many symbolic models of cryptographic primitives date back to the early 2000s and the foundations of the field, for example the most commonly used models of symmetric encryption, hashing, signing and public key encryption date back to early work on ProVerif. Further work has extended the range of primitives covered to include Diffie-Hellman groups, lists and counters [12].

However, newer protocols are using an increasingly wide range of cryptographic primitives which do not fall into the existing models, for example, verifiable random functions, homomorphic encryption schemes or post-quantum signatures. Without accurate and well justified models of these primitives, it is not possible to meaningfully analyze protocols which use said primitives. This thesis focuses on modelling homomorphic encryption.

2 Overview

Most symbolic systems, among them Tamarin, are based on equational reasoning. Their strategy is to reduce security problems to constraint solving problems in a term algebra and exhaustively searching all possible states that represent executions of the protocol. To model different cryptographic primitives, the term algebra is defined modulo various (possibly user-defined) equational theories, expressed as rewrite systems. For example, it is possible to model encryption and decryption operations by representing encryption and decryption with abstract functions symbols enc and dec that satisfy the following rewrite rule:

$$dec(enc(X, K), K) \rightarrow X.$$

During the exhaustive search of possible protocol executions Tamarin performs unification of different terms in the term algebra. Tamarin uses a technique called *folding variant narrowing*, a general-purpose procedure that applies to a broad sample of equational theories, allowing the user to choose which cryptographic primitives he wishes to model and how.

However this algorithm works only if the theories satisfy a certain property, the finite variant property (FVP). The objective of this project is to extend Tamarin to support theories that do *not* fall in this category, under the assumption that there is an existing specialized unification algorithm. In particular, the equational theory given by the following rewrite rule

$$enc(X \star Y, K) \rightarrow enc(X, K) \star enc(Y, K), \tag{1}$$

representing homomorphic encryption, does not obey the FVP, regardless of whether the operator $\star$ has additional algebraic properties or not.

For such primitives, a clear alternative is to implement a specialized unification algorithm. This specialized algorithm however needs to be combined with the approach currently used by Tamarin.

In this thesis, we will consider the particular case of the homomorphic encryption rule (1), where the $\star$ operator does not have any additional algebraic properties.

The paper [2] proposes a specialized algorithm that performs unification modulo the theory of homomorphic encryption (1). This algorithm has later been implemented in the tool Maude-NPA [9].

The goal of this thesis is to integrate this algorithm also in the Tamarin prover, by combining this algorithm with the existing one used by Tamarin.

3 Prerequisites

As this thesis is focused on the implementation, the student must be familiar with Haskell (Tamarin's codebase is written in Haskell) and interested in improving their functional programming skills.

4 Assignment

4.1 Objective

The objective of the proposed thesis work is the following: Implement an existing unification algorithm for an equational theory describing homomorphic encryption in the Tamarin prover and combine it with Tamarin's existing unification approach.

More precisely, the goal of this thesis is threefold:

- Implement the unification algorithm modulo homomorphic encryption described in [2].
- Integrate this algorithm with Tamarin's current unification approach, assuming that the defined equational theories have *disjoint* signature (see [3]) .
- Investigate possible unification algorithms for the theory of homomorphic encryption over an operator that satisfies additional algebraic properties (stretch goal).

4.2 Tasks

We suggest the following initial breakdown of the task into subtasks, subject to refinement in the student's time schedule.

Part 1:

1. Understand Tamarin's unification approach and get familiar with the Tamarin codebase (which is written in Haskell).
2. Study the unification algorithm proposed in [2].
3. Implement this algorithm in the Tamarin prover assuming this is the only equational theory used.
4. Evaluate effectiveness of approach on a (artificial or not) case study.

Part 2:

1. Review literature on combining unification algorithms for the union of disjoint equational theories [3].
2. Implement such an integration in the Tamarin prover. In particular this will require combining the unification algorithm from [2] already implemented in the previous phase with Maude's unification algorithm for the AC equational theory (associativity and commutativity).
3. Evaluate effectiveness of approach on a (artificial or not) case study.

Part 3:

1. If time permits, investigate a possible unification algorithm for the theory of homomorphic encryption over an operator that has additional properties (such as an abelian group operator).

Depending on the student's interest and progress, there is considerable flexibility as to which Part 3, considering additional properties, will be tackled (it is not an easy problem).

4.3 Relevant protocols

We give some examples of relevant protocols that use homomorphic encryption (currently out of Tamarin's scope) which could be used to evaluate the implementation. In particular [7] gives a survey of protocols using homomorphic encryption.

- **Block ciphers.** ECB mode of encryption divides the message into blocks and encrypts each block separately. This can be seen as homomorphic encryption over the concatenation operator (encrypting a concatenation of blocks equals the concatenation of encryption of blocks). A first sanity check would consist in modelling such block cipher abstractly and confirming its known vulnerabilities. For example an easy case study would be using encryption in Electronic Code Book (ECB) mode for the Needham-Schroeder-Lowe protocol (see [7]).
- **E-voting protocols.** Many e-voting protocols use homomorphic encryption to ensure vote privacy and would be interesting to model. Examples of such protocols are Helios 2.0[1] or Civitas [6].

4.4 Deliverables

1. At the end of the second week of the thesis a provisional time schedule for the master thesis must be given, which is then discussed and refined together with the supervisor. Regular meetings are expected to be held between the supervisor(s) and the student.

2. At the end of the master thesis, a presentation of 30 minutes, with additional time for questions, must be given. It should give an overview as well as highlight the most important details of the work.

3. The final report must be written in English and typeset in LaTeX. It should include an introduction and motivation, an overview of the related work, and a detailed description of the obtained results. Three copies of the final report must be delivered to the supervisor.

4. All developed code, including Tamarin models, as well as all case studies, must be available as source code that compiles with a standard build environment.

5. All source code developed during the thesis (including the LaTeXcode for the final report) must be checked into a version-controlled repository managed by the InfSec group. This code may be used by the InfSec group in any form for all purposes the group sees fit provided appropriate credit to the student is given.

References

[1] B. Adida. Helios: Web-based open-audit voting. In *Usenix Security Symposium*, pages 335–348, 2008.

[2] Siva Anantharaman, Hai Lin, Christopher Lynch, Paliath Narendran, and Michael Rusinowitch. Cap unification: application to protocol security modulo homomorphic encryption. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10. Association for Computing Machinery, 2010.

[3] Franz Baader and Klaus U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *Journal of Symbolic Computation*, 21(2):211–243, 1996.

[4] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. A formal analysis of 5g authentication. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1383–1396, 2018.

[5] B. Blanchet. Proverif automatic cryptographic protocol verifier user manual. *CNRS, Departement dInformatique, Ecole Normale Superieure, Paris*, 2005.

[6] M.R. Clarkson, S. Chong, and A.C. Myers. Civitas: Toward a secure voting system. In *S&P: Proc. Symposium on Security and Privacy*, 2008.

[7] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. In *Journal of Computer Security*, pages 1–42, 2006.

[8] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of tls 1.3. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1773–1788, 2017.

[9] Santiago Escobar, Deepak Kapur, Christopher Lynch, Catherine Meadows, José Meseguer, Paliath Narendran, and Ralf Sasse. Protocol analysis in maude-npa using unification modulo homomorphic encryption. In *Proceedings of the 13th International ACM SIGPLAN Symposium on Principles and Practices of Declarative Programming*, PPDP '11. Association for Computing Machinery, 2011.

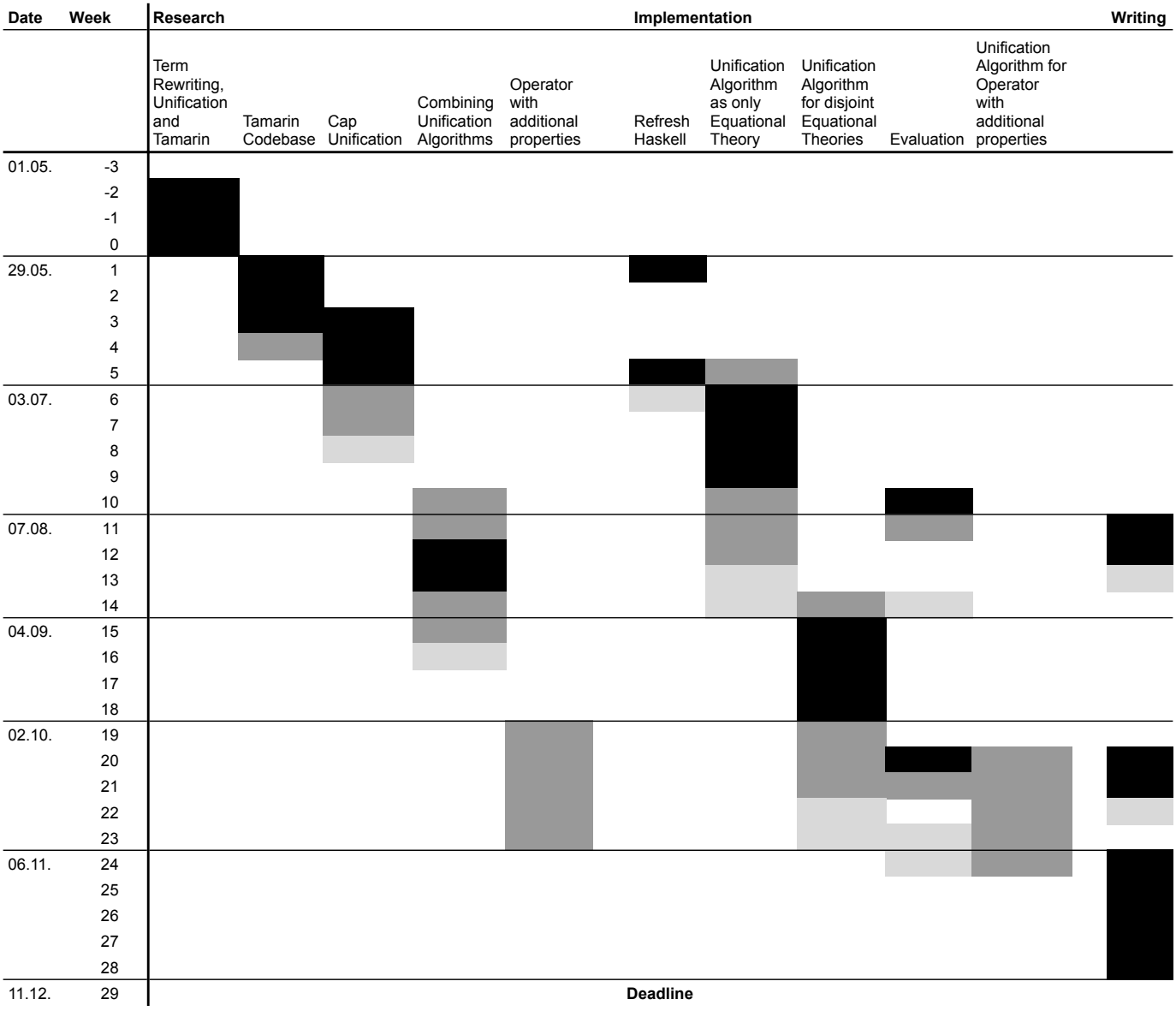
[10] Guillaume Girol, Lucca Hirschi, Ralf Sasse, Dennis Jackson, Cas Cremers, and David Basin. A spectral analysis of noise: A comprehensive, automated, formal analysis of diffie-hellman protocols. In *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

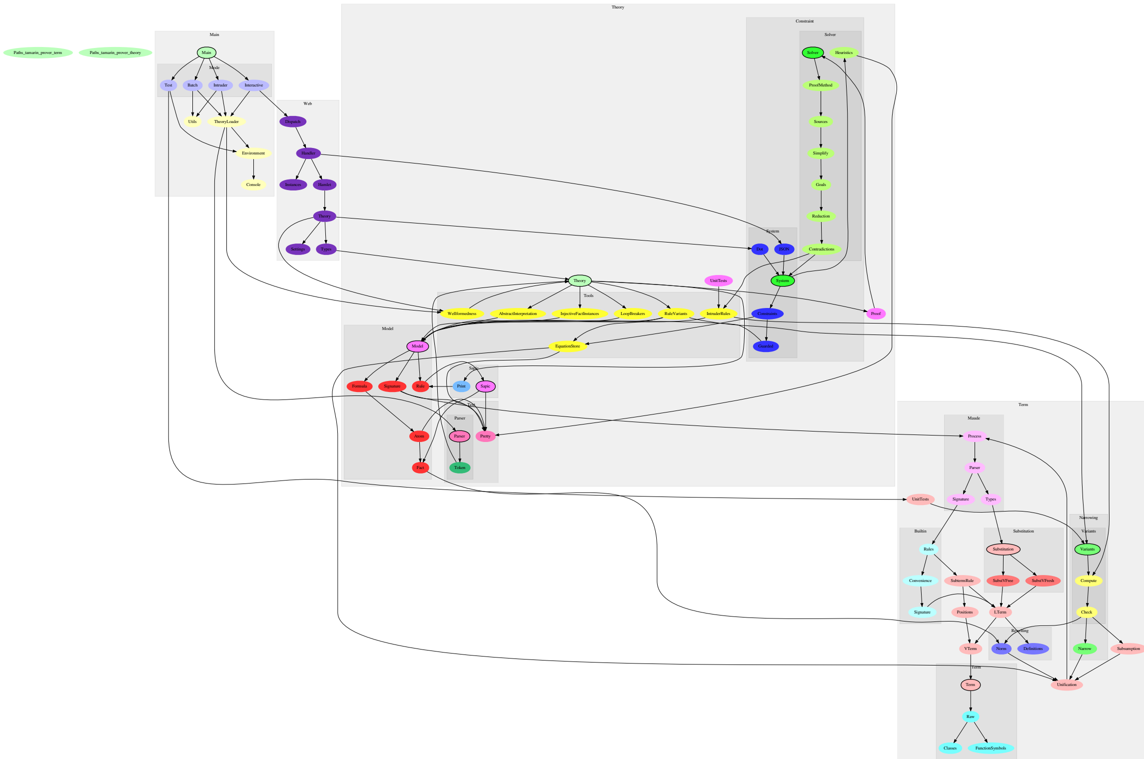
[11] S. Meier, B. Schmidt, C. Cremers, and D. Basin. The Tamarin Prover for the Symbolic Analysis of Security Protocols. In *Proc. 25th International Conference on Computer Aided Verification*, volume 8044 of *LNCS*, pages 696–701. Springer, 2013.

[12] Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In Stephen Chong, editor, *CSF*, pages 78–94. IEEE, 2012.

Master's thesis provisional time schedule

Unification modulo homomorphic encryption in the Tamarin Prover — Jonas Passweg




Paths_tamarin_prover_term
Paths_tamarin_prover_theory
Main
Main
Mode
Test
Batch
Intruder
Interactive
Utils
TheoryLoader
Environment
Console
Web
Dispatch
Handler
Instances
Hamlet
Theory
Settings
Types
Theory
Constraint
Solver
Solver
Heuristics
ProofMethod
Sources
Simplify
Goals
Reduction
Contradictions
System
Dot
JSON
System
Constraints
Guarded
UnitTests
Theory
Tools
Wellformedness
AbstractInterpretation
InjectiveFactInstances
LoopBreakers
RuleVariants
IntruderRules
EquationStore
Proof
Model
Model
Formula
Signature
Rule
Sapic
Print
Sapic
Text
Parser
Parser
Pretty
Token
Atom
Fact
Term
Maude
Process
Parser
Signature
Types
UnitTests
Builtin
Rules
Convenience
Signature
SubtermRule
Substitution
Substitution
SubstVFree
SubstVFresh
Positions
LTerm
Narrowing
Variants
Variants
Compute
Check
Rewriting
Norm
Definitions
VTerm
Narrow
Subsumption
Term
Term
Unification
Raw
Classes
FunctionSymbols