Human Errors and Natural Strategies

Benjamin Bordais[1], Wojtek Jamroga[2], and Damian Kurpiewski[2]

[1] ENS Rennes
[2] Institute of Computer Science, Polish Academy of Sciences

Abstract. Formal verification of security and voting protocols is a very active field of research. However, it is very often assumed that every agent involved in a protocol follows accurately each step of that protocol, which ignores the nondeterminism that could arise from misbehavior. An interesting attempt to include this possibility was presented in [3] by *Basin et al.* where the human agents involved in security protocols are now able to have an erratic behavior. Then, the authors designed simple constraints that, if followed by the human agents, would ensure the correctness of the protocol. In this study, we look at a way to generate an appropriate set of constraints from the shape of the protocol of interest, instead of applying the same simple constraints to every protocol. Specifically, the contribution is threefold. Firstly, we apply the methodology of [3] to include human errors in voting protocols . Secondly, we look at more elaborate set of constraints that can be semi-automatically generated from the protocols of interest. These constraints take the form of natural strategies (that is, strategies meant for humans, it was introduced in [9]). Thirdly, in order to find the appropriate set of constraints, we implemented a systematic search through the space of available strategies. To formally verify the correctness of the protocols, we used the theorem prover Tamarin that was also used in [3].

1 Introduction

Cryptographic protocols, whether they are security protocols or voting protocols, are widely used on a daily basis. For instance, an e-banking transaction or a vote in an electronic election require the use of such protocols. Since these protocols are so much used, we need to have a formal guarantee that these protocols ensure some desired properties like Message-Authentication for security protocols or Vote-Privacy for voting protocols. See for instance [5] for the formal proof that the voting protocol Selene [14] ensures Vote-Privacy and Receipt-Freeness.

However, very often, humans are the ones using these protocols which leads to the possibility that some attacks that exploit the fallibility of human are enabled even if the protocol in itself (that is, when humans act as they are supposed to) is correct. It is very possible that humans may not know exactly what they are supposed to do or to carelessly forget some protocol steps. Human weakness is very well exemplified in the effectiveness of phishing websites, even when humans are informed in a lab environment [1,7]. Hence, the necessity to include humans in the study of crpytographic protocols.

In [3], *Basin et al.* tackled that issue by studying the impact of introducing human errors in security protocols whose correctness (without considering humans) had previously been proven. Specifically, they looked at whether or not the correctness would still hold with human errors. Since it was not the case for several protocols, they designed generic rules so that, if they were correctly followed, it could restore the correctness of the protocols. These rules were effective for every protocol but one. In any case, these rules gave an indication to the authors about the possible falws in the security protocols considered. All the results were established with the theorem prover Tamarin [12].

In this work, we use their way of defining and implementing human errors but we went in another direction to find the rules for the human to follow that would restore the correctness of the model. Our idea was to adapt the rules to the protocol of interest, and, ideally, to automatically generate them. In order to do that, we needed to know in which set to look for the right rules. We used an idea that was developed in [9], in the context of multi-agent system. The idea of the authors was to define what they called *natural strategies*, which are strategies designed to be understandable by humans. When translating that idea to the context of cryptographic protocol, we came up with strategies that were both simple enough and of bounded size. Since the authors of [3] used Tamarin, we did the same to establish all our results.

Main contributions Our main contribution first consists in extending the notion of human errors to two voting protocols: SimpleVote[3] and SimpleSelene[4]. Then, the tool we implemented is an OCaml program that takes a Tamarin model of a specific procedure as an input and searches for a successful natural strategy. To do so, we designed the shape of the strategies that would be adapted to each protocol and we found an appropriate way to explore the set of available strategies. The sum up of our results is presented in Table 1. Our tool was able to find an appropriate strategy for every protocol (even MPAuth_MA, the protocol for which the rules defined in [3] were not enough) but SimpleSelene for which Tamarin was not able to conclude in reasonable time on a single file (however we were able to use the interactive mode of Tamarin to prove that the correctness of the model does not hold with untrained humans), which implies that our tool cannot be used on that protocol. More detailed results can be found in section 6.

Related Works There are other works that tackle the issues that could arise when dealing with humans in the context of formal verification. In [13], *Rukšėnas et al.* allow human fallibility in the context of an interface between a human agent and a system. In that work, they do not consider any cryptographic protocol but they look at interactions between a human, the system and an adversary.

In [4], *Beckert et al.* work on a similar setting: they look at an interface between humans and machines. They use a psychological model for humans

[3] A simple voting protocol designed to test our tool.

[4] A simplified version of the implementation of the Selene voting protocol done in [5] in Tamarin that includes human errors.

Protocol	Untrained Humans	Rule Based Humans	Strategic Humans
Cronto_EA	✓	–	–
Cronto_MA	✗	✓	✓
Google2Step_EA	✓	–	–
Google2Step_MA	✗	✓	✓
MPAuth_EA	✗	✓	✓
MPAuth_MA	✗	✗[1]	✓
MPAuth_VC	✗	✓	✓
SimpleVote	✗	✓	✓
SimpleSelene	✗[2]	?	?

[1]The authors of [3] proved that no combination of the available rules could lead to the correctness of the protocol.
[2]This was proven by using the interactive mode of Tamarin.

Fig. 1: The correctness of security protocols (from Cronto_EA to MPAuth_VC) and of voting protocols (SimpleVote and SimpleSelene). Untrained humans are fallible human who do not follow any rule. Rule based humans follow the rules defined in [3]. Strategic humans follow a strategy that was generated and selected among all strategies available by the tool we implemented.

that is a variant of the GOMS model [10]. In their model, they explicitly take into account the possibility that humans do not behave as they should.

As for reasoning about strategic ability, apart from [9] that deals with strategies simple enough to be practically carried out by humans and their inherent mental operations, they are other papers considering humans with limited possibilities. See for instance [11] where the effect of bounded memory for players in games is studied. However, this approach is entirely mathematical without any grasp on the practical aspect of decision making.

Finally, in all this work, we use the Tamarin theorem prover that allows verification of cryptographic protocols. See [12] for an overview of the tool and its applications.

Structure of the paper Sect. 2 introduces how to define a protocol in Tamarin. Sect. 3 defines the interesting properties we want cryptographic protocols to ensure. Sect. 4 shows how we can introduce human errors in a protocol. Sect. 5 considers the way we define our strategies. Finally, Sect. 6 covers the implementation and results we got.

2 Security and Voting protocols in Tamarin

In this section, we present how to model a protocol (we do not make a distinction between a voting and a security protocol for now).

2.1 Messages in Tamarin

The messages that are exchanged in a protocol follow a specific term algebra. A message is a term which is either a variable or a function applied to other terms. Some classical functions are available in Tamarin such as the pairing function $<$, $>$, or the signing function sign(,). Some cryptographic functions are also available. For instance, the symmetric encryption senc and the symmetric decryption sdec can be used in Tamarin. The variables can either be taken from a infinite set of fresh constants (often preceded by ˜) or an infinite set of public constants (often preceded by $). The equality between messages is modulo an equational theory. For example, in the case of symmetric encryption, we have the equality sdec(senc(m,k),k) $=$ m. The details of this equational theory can be found in [2].

```
rule R1:
  [ ]
  --[ L('x') ]->
  [ F('1','x'), F('2','y') ]

rule R2:
  [ F(u,v) ]
  --[ M(u,v) ]->
  [ H(u), G('3',h(v)) ]
```

Fig. 2: Two rules defining a protocol P in Tamarin.

$$\begin{tikzcd}
\text{[]} \arrow[d] \\
\text{[F('1','x'), F('2','y')]} \arrow[d] \\
\text{[F('2','y'), H('1'), G('3',h('x'))]} \arrow[d] \\
\text{[H('1'), G('3',h('x')),H('2'), G('3',h('y'))]}
\end{tikzcd}$$

Fig. 3: A possible execution of the protocol P.

2.2 Specifying a protocol in Tamarin

The execution model of Tamarin is a multiset term-rewriting system. A state of the system is a multiset of facts. A fact is either linear or persistent. A linear fact can be added to the state of the system, and it can also be removed. However, a persistent fact can only be added. A persistent fact's symbol is preceded by an exclamation point. The initial state of the system is the empty multiset. The way facts are added and removed to the system state is defined by transition rules of the following shape, **rule** Example: [{Consumed}] −−[{Actions}]−> [{Generated}]. For such a rule to be applied, the system state must contain every fact in {Consumed}. Then, the consumed linear facts are removed from the system state and the facts in {Generated} are added. A trace is a finite sequence of multiset of actions that come from the application of transition rules. In the case of the rule Example, the multiset {Actions} is stored in the trace. Some rules are initially defined in Tamarin. For instance, the fresh rule []−−[]−>[Fr(˜x)] produces the fact Fr(˜x). Moreover, the rules defining the possibilities of the adversary are also defined. The adversary (or attacker) follows the classical Dolev-Yao model [8]. That is, the adversary can intercept any message that is exchanged on the insecure channel (that is a message that is outputted with the fact Out()). The adversary can generate fresh constants, use public ones and construct any message with these and public functions. Then, he can send it to the insecure channel (which generates the fact In()). A protocol P consists in a set or rules. We denote by $Traces(P)$ the set of all traces that can occur in P. In Figure 2, a protocol P is defined by two rules in Tamarin. The facts are in green, and the actions are in red. Then, in Figure 3, we see a possible execution that could arise from protocol P, where R1 is executed once, and then rule R2 is executed twice. The trace corresponding to that execution can be seen in Figure 4.

$$Traces(P) \ni tr = \{\mathsf{L('x')}\}\ \mathsf{@i},\ \{\mathsf{M('1','x')}\}\ \mathsf{@j},\ \{\mathsf{M('2','y')}\}\ \mathsf{@k},$$
$$i < j < k$$

Fig. 4: The corresponding trace to the execution of the protocol P. i, j and k are ordered timepoints

Defining a secure channel in Tamarin In Tamarin, messages can be outputted by using the fact Out(), and then the adversary has access to these messages and can modify what is received. However, it is sometimes required in a protocol to use a secure channel, that is a channel that is not accessible to the adversary. Specifically, the adversary cannot see any message exchanged and he is unable to send any message of his own on this kind of channel. In [2], the authors define the HISP model extension of Tamarin, in which secure channel are defined. A fact Out_S($A,$B,xn,x) is generated in a rule if agent $A sends the message s of type xn to agent $B. The use of In_S($A,$B,xn,x) is similar. The implementation in Tamarin of secure channel (as well as other kind of channels) can be seen in appendix A.

3 Desired properties

Up until now, we have not made any distinction between a security and a voting protocol. However, there is an inherent difference between these two kinds of protocol. A voting protocol contains far more non determinism than a security protocol. For instance, the voters can choose their candidates, and they can choose if and when they want to try to verify their vote. Whereas in a security protocol, the role of each agent is specified so that there is not much choice left for the agent. It has to be noted that this difference fades away when these protocols are modeled in Tamarin, since the non determinism of a voting protocol is almost entirely resolved in the way the protocol is implemented. Nonetheless, there is still a difference between a security and a voting protocol in the properties that these protocols should ensure.

3.1 Security protocol

The properties of interest for the security protocols are the ones that were defined in [3], where the formal definition of each of these properties can be found.

Generally speaking these properties deal with authentication. Let us consider, for instance, the property of Message authentication. We have Message authentication if a given agent is sure that another agent sent a certain message. More specifically, a protocol ensures Message authentication for an agent A to an agent B if whenever the agent B commits to a message m, then agent A has previously sent that message m. The properties of Device or Entity authentication are also defined.

For every security protocol, we also require that a functionality property holds. That is, we require that it is possible that the protocol is executed up to its very end. For instance, in the previous example where Message authentication has to be ensured, it has to be possible for an agent B to commit to a message m.

3.2 Voting protocol

In the context of voting protocols, the most common property is Vote-Privacy. It requires that the protocol does not disclose the vote of any of the voter. In [6], the authors defined that property to be an equivalence in the possibilities of the adversary between two cases where two voters choose a candidate A and B respectively and then swap their votes. This is the usual way that this property is verified. In Tamarin, it can be verified by using observational equivalence with the keyword diff. This technique is also the one used to verify the property Receipt-Freeness. This is another very classical property to verify in voting protocols, it specifies that a voter cannot prove to a coercer that he has voted in a particular way. However, when Tamarin is running on a protocol that uses the keyword diff, it takes a lot of time to conclude. Since our tool is supposed to launch Tamarin on a lot of different files, it is not viable to have properties that take that long to verify. Therefore, we do not consider that property. However, we do require that our protocols ensure Verifiability. This means that whenever a voter has voted and has verified his vote, then what he gets is the vote he intended to cast.

As for the security protocols, we also require that the protocol can be executed up to its very end. In this case, we require that it is possible for every voter to verify his vote.

3.3 Using Tamarin to verify these properties

In Tamarin, we can define lemmas that express the properties we mentioned earlier. A lemma is defined in the following way in Tamarin : **lemma** L: Q "F" where Q is a quantifier over traces, either equal to **exists−trace** (then, we have an existential lemma) or it can be equal to **all−traces** (in that case, we have a universal lemma). F is a guarded first order formula over actions (that appear on the traces) with a sort on timepoints. We denote by $Traces$(F) the set of traces that comply with the formula F. In Figure 5, it is shown how some of the properties we discussed earlier can be expressed with lemmas in Tamarin.

```
lemma Verifiability: all−traces
"All #i v id. Verified(<id,v>)@i ==> (Ex #j. Vote(v) @j & j < i)"

lemma Functionality: exists−trace
"Ex #i id v. Verified(<id,v>)@i"
```

Fig. 5: Lemmas in Tamarin that express Verifiability and Functionality.

The universal lemma Verifiability expresses that whenever an action Verified(<id,v>) appears on a trace (which corresponds to the verification by a voter that he cast the vote v), then that voter has previously (since the timepoint of the action Verified (resp. Vote) is j (resp. i), and we have j < i) voted for v. A protocol P ensures that lemma if the corresponding first order formula is verified by every trace in $Traces(P)$.

The lemma Functionality is a little bit different since it requires that at some point, the action Verified(<v,i>) occurs. That lemma expresses the possibility to execute the protocol up to its very end. A protocol P ensures that lemma if the corresponding first order formula is by verified at least one trace in $Traces(P)$.

Let us define formally at which condition a protocol P ensures a lemma L (denoted $P \models$ L). Let **lemma** L1: **exists−trace** "F" and **lemma** L2: **all−traces** "F" be an existential and a universal lemma. Then, the protocol P ensures:

- L1 if and only if $Traces(P) \cap Traces(\mathsf{F1}) \neq \emptyset$;
- L2 if and only if $Traces(P) \subseteq Traces(\mathsf{F2})$.

In Tamarin, there are two ways to establish a proof. We can either use the fully automated mode, that makes use of deductions and equational theory reasoning and possibly some heuristics. The output of that mode is either that the lemma hold with a proof of correctness, or a counterexample, that is a trace that does not verify the first order formula expressed defining the lemma (it is possible to have more than one lemma defined for a single protocol, in which case there is one output per lemma). However, since the verification problems Tamarin deals with are not decidable, it is possible that Tamarin does not terminate. We can also use the interactive mode of Tamarin to combine manual and automated proof. When we mention the time Tamarin takes to conclude, we always refer to the fully automated mode.

4 Human Errors

4.1 Defining human Errors

We use the definition of fallible humans defined in [3]. Fallible humans are not able to perform any complex operation. They are represented with their knowledge that can be updated according to what is received on some channels (whether it is secure or insecure). They can also use their knowledge to send messages. However, they are not able to encrypt any message by themselves. These possibilities adds non determinism in the protocol. In [3], fallible humans who do not follow any rule are called untrained humans. In contrast, fallible humans who follow the simple rules that were designed are called rule based.

In contrast, human who only follow the protocol specifications are called infallible. In [3], fallible humans who follow the simple rules that were designed are called rule based.

4.2 Human errors in Tamarin

Some additional rules are added to the protocol implemented in Tamarin to take into account the fallibility of humans. The complete set of rule defining human behavior can be found in appendix B. In Figure 6, we can see how fallible human

```
rule H_send_S:                                   rule H_receive_S:
  [ !HK($H,$x.1,x.2) ]                             [ In_S( $A,$H, $x.1, x.2 ) ]
  --[ Send($H,$x.1,x.2), H($H), To($A)]->          --[ Receive($H,$x.1,x.2),H($H), From($A)]->
  [ Out_S($H,$A,$x.1,x.2) ]                        [ !HK($H,$x.1,x.2) ]
```

Fig. 6: Some rules defining the exchange of messages on secure channel for fallible humans in Tamarin

exchange messages on a secure channel. The fact !HK($H,$x.1,x.2) expresses that the human agent knows x.2 which is of public type $x.1 (note that since that fact is persistent, it will always be available). The rule H_send_S gives the possibility to send something that the human knows on a secure channel. The action H($H) is only here to specify that this is a rule describing human's behavior . Then, the action Send($H,$x.1,x.2) appears with the action To($A), that is here because we are dealing with a secure channel (thus, the identity of the receiver is known). The rule H_receive_S is quite similar, and describes the update of the knowledge of the human from a message received on a secure channel. The same kind of rules exist for the insecure channel, however, in that case, the actions To($A) and From($A) do not appear (since the human could be exchanging messages with the attacker). Human agents have also the possibility to generate a fresh constant and add it to their knowledge.

5 Natural Strategies

Once human errors is introduced in a protocol, it often follows that the correctness of that protocol does not hold anymore. Therefore, in order to restore it, we need to limit the possible behaviors of human agents. However, these limitations has to be in compliance with the human capability of following rules.

Reasoning about strategies simple enough to be followed by humans was done in [9] in the context of multi-agent systems. In such a context, a strategy for an agent is a function that associates to a state (or a sequence of states) of the system an action to execute. It can be seen as a conditional plan for the agent. This approach is well suited in the case where agents are machines but it is not appropriate for reasoning about human behavior. The idea of this paper is that when considering the formula defining strategies, we only look at bounded-size formula. This allows to reason about strategies in which the conditions specifying which action to execute are not too complicated. They are called natural strategies.

We use the same idea in the context of human errors in cryptographic protocols. In our case, the interest of using simple and bounded-size strategies is twofold: first, it is understandable and usable by humans, second, since we want to automatically generate strategies, we need to consider a bound on the complexity of the strategy we consider.

5.1 Strategies in Tamarin

Before looking how simple, and size-bounded strategies translate in our context, we need to see what constructions are allowed in Tamarin to limit the behavior of human agents. In Tamarin, we can use the keyword **restriction** so that some traces are not considered in a protocol. A restriction is defined similarly to a lemma except that it does not have a quantification over traces since it necessarily concerns every traces. By r(F), we denote **restriction** r: F for a first order formula over actions F. Then, if we have a protocol P and a formula F, the set of traces that will be considered for ensuring lemmas for that protocol P with the restriction r(F) will be $Traces(P) \cap Traces(\mathsf{F})$. That is, only the traces that satisfy the formula F are considered.

Simple strategies In order to have restrictions understandable by humans, we only consider restrictions with one logical implication and at most two actions in both side of the implication. We use two kinds of restriction: *channel restrictions* and *type restrictions*. A channel restriction forbids sending or receiving message on an insecure channel with a given type of message. A type restriction specifies a condition to receive (or send) a message of a given type on a secure channel. For instance, we can have these restrictions:

restriction channel: "**All** #k H m.Send(H,'pw',m)@k **==>** **Ex** S.To(S)@k"

restriction type: "**All** #k H m S. Receive(H,'m',m)@k **&** From(S)@k
==> **Ex** #j. Send(H,'m',m)@j **&** To(S)@j **&** j<k"

This channel restriction forbids to send a message of type 'pw' (for password) on an insecure channel. That type restriction forbids one to receive, on a secure channel, a message of type 'm' if one did not send that same message previously on a secure channel. Every restriction we consider is similar to one of these two, with potentially reversing the role of Send and Receive (and also of To and From). When human agents follow a strategy, they are called *strategic humans*.

Bounded-size strategies A strategy is a set of restrictions. We fix the maximum number of restrictions in a strategy to be 3. It is a compromise between having an interesting (that is, large) set of strategies and having a set of strategies small enough so that it does not take days to find an appropriate strategy. In a strategy several restrictions may be combined, but we tried not to have too similar restrictions in the same strategy. As a strategy is a combination of several restrictions, it can be seen as a restriction whose formula is conjunction of at most three other formulas. For a strategy $\mathsf{s} = \mathsf{r}(\mathsf{F})$ and a protocol P, we denote by $P^{[s]}$ the protocol that contains all the traces of P but the traces that do not comply with the strategy s. That is, $Traces(P^{[s]}) = Traces(P) \cap Traces(\mathsf{F})$.

Differences with the rules defined in [3] These rules (which are restrictions) can be found in appendix C. They are quite similar to our restrictions, even if they are a little more elaborate (and therefore a little less understandable). Our restrictions are more 'powerful', in the sense that we can restore the correctness of the model MPAuth_MA, which is not possible with their rules. Overall, the main difference lies in what we do with them: we try to combine them to find an appropriate strategy, whereas *Basin et al.* manually test them one by one.

5.2 Properties of the strategies

Let us consider two first order formulas F1 and F2 such that $\mathsf{F1} \Rightarrow \mathsf{F2}$ (for instance, $\mathsf{F1} = \mathsf{FA}\ \&\ \mathsf{FB}$ and $\mathsf{F2} = \mathsf{FA}$ on a protocol P. Then, if we consider the two strategies $\mathsf{s1} = \mathsf{r}(\mathsf{F1})$ and $\mathsf{s2} = \mathsf{r}(\mathsf{F2})$, we have $Traces(P^{[s1]}) \subseteq Traces(P^{[s2]})$. Consider two lemmas **lemma** L1: **exists−trace** "G1" and **lemma** L2: **all−traces** "G2". Let us assume that Tamarin proves that $P^{[s2]} \not\models L1$. Therefore, according to Section 3.3, we have $Traces(P^{[s2]}) \cap Traces(\mathsf{G1}) = \emptyset$. It follows that $Traces(P^{[s1]}) \cap Traces(\mathsf{G1}) \subseteq Traces(P^{[s2]}) \cap Traces(\mathsf{G1}) = \emptyset$. Hence, $P^{[s1]} \not\models L1$. Symmetrically, we can prove that if $P^{[s1]} \not\models L2$, then $P^{[s2]} \not\models L2$. These observations allow us to deduce that a strategy does not ensure a lemma without having to launch Tamarin on file corresponding to that strategy.

6 Contribution

Implementation We implemented an OCaml program that is able to extract from a protocol written in Tamarin every type of messages that appears in that protocol. Then, one is able to decide which restrictions will be considered (channel restrictions, type restrictions, or both) and what types of messages will be used to generate strategies. The program can then explore that set of possible strategies. All the code can be found at `https://github.com/blackbat13/nsverif`. In order to properly use the observations made in the previous section, the program also associates to each strategy s two sets s.more_traces and s.less_traces that contains strategies that are, respectively, less and more restrictive. In that way, if Tamarin proves that a strategy s does not ensure a universal (resp. existential) lemma, then neither does every strategy belonging to the set s.more_traces (resp. s.less_traces).

Algorithm Given a protocol and the restrictions of interest, we want to find a strategy built from those restrictions that restores the correctness of the protocol. We first try without any strategy (untrained humans). If at least one lemma is not verified, we launch Tamarin on every strategy that contains only one restriction. If we do not find an appropriate strategy among these, we try to find a strategy that is more elaborate. First, we associate a score (an integer) to every strategy tested. Second, we look into the sets s.less_traces in a decreasing order of the scores of the strategies s. Let us discuss how that score is computed. Because we look into the sets of type less_traces, if a strategy s does not ensure an existential lemma, no strategy in s.less_traces would work. The score of the strategy s is, in that case, 0. For a given strategy, if a universal lemma is not verified, we add n to its score, where n is the number of steps Tamarin took (usually between 5 and 500) to conclude[5] that it does not hold. If it is verified we add 100 000 [6] to the score. However, anytime Tamarin takes more time than a specific duration (a timeout) the computation is stopped and no result is outputted. In that case, the score attributed is equal to 10 000 so that the corresponding strategy is chosen before any strategy that do not ensure any universal lemma but is chosen after a strategy that ensures at least one universal lemma. The order of the strategy that are tested in the sets s.less_traces of interest random. Once a strategy s is proven to be correct, we search into the set s.more_traces for a less restrictive strategy that is still correct.

6.1 Result

The summary of our results can be found in Table 1. As we can see, our program is able to find a correct strategy for every protocol but SimpleSelene. Moreover, on the security protocol MPAuth_MA, the rules defined in [3] were not enough to restore correctness, whereas it is the case of our strategies. However, it has to be noted that the rules defined in [3] give a positive result for the voting protocol SimpleVote (they were designed for security protocols). As for the voting protocol SimpleSelene, Tamarin takes too much time on the fully automated mode to get any result. That table lacks computing times. They are given in Table 7.

Overall, the time taken by our program is much higher that the time taken for the rule based human case. However, this was predictable since our program launches Tamarin on a lot of files to find an appropriate strategy. For the first three protocols, an appropriate strategy is found among the ones with only one restrictions, therefore no random is used. However, this is not the case for the three other protocols that do use random, therefore the results are given as statistics on 10 attempts. We can see that there are a lot variations with the random seed used in the protocol, which is best exemplified by the difference between the minimum and maximum time taken to conclude. However, even in

[5] That choice is arbitrary. The idea is that the more steps Tamarin takes to conclude, the less obvious it is that the lemma does not hold. However, since the way Tamarin proves or disproves a lemma is not straightforward, that choice is questionable.

[6] Chosen to be much higher that any number of steps taken by Tamarin.

the worst case scenario, the time taken is still reasonable since these kind of computations only have to be done once per protocol.

For every security protocols, both channel and type restrictions were used. For the voting protocol SimpleVote, only channel restrictions were used. In both cases, there were two types of the messages appearing on the restrictions. The time taken by our program depends on the timeout that was set. The choice of that timeout is quite difficult and we fixed our choices after a few try. The

Protocol	RBH	Timeout	Mean	Standard deviation	Minimum	Maximum
Cronto_MA	2.7	100.0	191.1	–	–	–
Google2Step_MA	2.1	30.0	17.8	–	–	–
MPAuth_EA	158.1	100.0	340.2	–	–	–
MPAuth_MA	–	100.0	1969.8	673.3	604.5	3000.7
MPAuth_VC	7.8	240.0	5753.4	2434.9	1473.5	8506.8
SimpleVote	3.1	30.0	55.75	5.8	48.3	68.0

Fig. 7: The time taken by our program to find a correct strategy. All the time considered here are in seconds. Experiments done on a personal computer with processor Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz, Ubuntu 18.04, 8GB of RAM

effectivness of our exploration of the set of strategies is studied in appendix D.

6.2 The SimpleVote protocol

We did not get any result on the voting protocol SimpleSelene which is an adaptation of the voting protocol Selene [14]. One of the reason for that may be that, in that protocol, the messages that are exchanged consist in pairs of messages. That is why we defined the voting protocol SimpleVote. In that protocol, agents do not exchange pairs of messages anymore. However, we made them use both secure and insecure channels so that it is not straightforward to find an appropriate strategy. As it initially was, we had the same problem that we had with SimpleSelene: Tamarin could not conclude in reasonable time. To simplify the protocol, we removed the possibility for human agents to receive messages on a secure channel. In the way the protocol was implemented, that does not impact whether or not the lemmas hold. However, Tamarin was then able to conclude in reasonable time (it is even quite fast, as we can see in Table 7).

6.3 Perspectives

The results we have show that, even if it takes quite some time, our idea to find an appropriate strategy by generating a lot of strategies is viable. The main goal of our work is to illustrate how it could be possible to find strategies suited for a specific requirement. The results we have allow us to think that our approach could be successful with other protocols, possibly in another context.

References

1. Alsharnouby, M., Alaca, F., Chiasson, S.: Why phishing still works: User strategies for combating phishing attacks. International Journal of Human-Computer Studies **82**, 69–82 (2015)
2. Basin, D., Radomirovic, S., Schläepfer, M.: A complete characterization of secure human-server communication. In: 2015 IEEE 28th Computer Security Foundations Symposium. pp. 199–213. IEEE (2015)
3. Basin, D.A., Radomirovic, S., Schmid, L.: Modeling human errors in security protocols. In: Proceedings of CSF. pp. 325–340 (2016). https://doi.org/10.1109/CSF.2016.30
4. Beckert, B., Beuster, G.: A method for formalizing, analyzing, and verifying secure user interfaces. In: International Conference on Formal Engineering Methods. pp. 55–73. Springer (2006)
5. Bruni, A., Drewsen, E., Schürmann, C.: Towards a mechanized proof of selene receipt-freeness and vote-privacy. In: Proceedings of E-Vote-ID. pp. 110–126 (2017). https://doi.org/10.1007/978-3-319-68687-5_7
6. Delaune, S., Kremer, S., Ryan, M.: Verifying privacy-type properties of electronic voting protocols: A taster. In: Towards Trustworthy Elections, pp. 289–309. Springer (2010)
7. Dhamija, R., Tygar, J.D., Hearst, M.: Why phishing works. In: Proceedings of the SIGCHI conference on Human Factors in computing systems. pp. 581–590. ACM (2006)
8. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on information theory **29**(2), 198–208 (1983)
9. Jamroga, W., Malvone, V., Murano, A.: Reasoning about natural strategic ability. In: Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS). pp. 714–722. IFAAMAS (2017)
10. John, B.E., Kieras, D.E.: The goms family of user interface analysis techniques: Comparison and contrast. ACM Transactions on Computer-Human Interaction (TOCHI) **3**(4), 320–351 (1996)
11. Kocherlakota, N.: Money is memory. Journal of Economic Theory **81**(2), 232–251 (1998). https://doi.org/https://doi.org/10.1006/jeth.1997.2357
12. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The tamarin prover for the symbolic analysis of security protocols. In: International Conference on Computer Aided Verification. pp. 696–701. Springer (2013)
13. Rukšėnas, R., Curzon, P., Blandford, A.: Modelling and analysing cognitive causes of security breaches. Innovations in Systems and Software Engineering **4**(2), 143–160 (2008)
14. Ryan, P., Rønne, P., Iovino, V.: Selene: Voting with transparent verifiability and coercion-mitigation. In: Financial Cryptography and Data Security: Proceedings of FC 2016. Revised Selected Papers. pp. 176–192 (2016)

A Alternative channels in Tamarin

Secure channel On a secure channel, the adversary can neither intercept any message nor send one of his. The implementation of a secure channel in Tamarin can be seen in Figure 8. The fact Sec is persistent because it is assumed that once a message is sent on a channel (whatever type of channel is considered), then it is possible to receive it any number of times.

```
rule ChanOut_S:                         rule ChanIn_S:
  [Out_S($A,$B,xn,x)]                     [!Sec($A,$B,xn,x)]
  --[ChanOut_S($A,$B,xn,x)]->             --[ChanIn_S($A,$B,xn,x)]->
  [!Sec($A,$B,xn,x)]                      [In_S($A,$B,xn,x)]
```

Fig. 8: An implementation in Tamarin of a secure channel

Authentic channel On an authentic channel, the adversary can learn the messages sent on the channel, but he can not modify it, nor modify the sender. The implementation of an authentic channel in Tamarin can be seen in Figure 9.

```
rule ChanOut_A:                         rule ChanIn_A:
  [Out_A($A,$B,xn,x)]                     [!Auth($A,xn,x), In($B)]
  --[ChanOut_A($A,$B,xn,x)]->             --[ChanIn_A($A,$B,xn,x)]->
  [!Auth($A,xn,x), Out(<$A,$B,xn,x>)]     [In_A($A,$B,xn,x)]
```

Fig. 9: An implementation in Tamarin of an authentic channel

Confidential channel On a confidential channel, the adversary cannot learn a message sent on the channel, however he can change the sender of a message or change the message that is sent. The implementation of a confidential channel in Tamarin can be seen in Figure 10.

```
rule ChanOut_C:                         rule ChanIn_C:
  [Out_C($A,$B,xn,x)]                   [!Conf($B,xn,x), In($A)]
  --[ChanOut_C($A,$B,xn,x)]->           --[ChanIn_C($A,$B,xn,x)]->
  [!Conf($B,xn,x)]                      [In_C($A,$B,xn,x)]

rule ChanIn_CAdv:
  [In(<$A,$B,xn,x>)]
  --[]->
  [In_C($A,$B,xn,x)]
```

Fig. 10: An implementation in Tamarin of a confidential channel

B Implementation of human errors in Tamarin

The rules defining the behavior of human agents can be seen in Figure 11. The rules H_receive and H_sned define the behavior of human agents dealing with insecure channel. Similarly, the rules H_receive_S and H_sned_S does the same thing with the secure channel. It has to be noted that, for these rules, actions To($A) and From($A) are generated. Finally, the rule H_fresh gives the possibility for human to learn a fresh nonce.

```
rule H_receive:
  [ In( <$x.1,x.2>) ]
  --[ Receive($H,$x.1,x.2),
  !HK($H,$x.1, x.2),H($H) ]->
  [ !HK($H,$x.1, x.2) ]

rule H_send:
  [ !HK($H,$x.1,x.2) ]
  --[ Send($H,$x.1,x.2),
  H($H) ]->
  [ Out(<$x.1,x.2>)]

rule H_send_S:
  [ !HK($H,$x.1,x.2) ]
  --[ Send($H,$x.1,x.2),
  H($H), To($A)]->
  [ Out_S($H,$A,$x.1,x.2) ]

rule H_receive_S:
  [ In_S( $A,$H, $x.1, x.2 ) ]
  --[ Receive($H,$x.1,x.2),
  !HK($H,$x.1,x.2), H($H), From($A)]->
  [ !HK($H,$x.1,x.2) ]

rule H_fresh:
  [ Fr(~x) ]
  --[ Fresh($H,$x.1,~x), !HK($H,$x.1,~x),
  H($H) ]->
  [ !HK($H,$x.1,~x)]
```

Fig. 11: The rules defining the human agents' behavior in Tamarin

C Rules defined in [3]

First rule The first rule defined in Tamarin can be found in Figure 12. For the rule to work, an action Rule3('Human','noTell',l) needs to appear on the traces of interest. If it does, a human agent cannot send any message of type l.

```
restriction noTell:
"All l m #s #i. Rule3('Human','noTell',l) @s & Send('Human',l,m) @i ==> F"
```

Fig. 12: The rule noTell defined in Tamarin

Second rule The second rule defined in Tamarin can be found in Figure 13. For the rule to work, an action Rule4('Human','noTellEx',l,lR) needs to appear on the traces of interest. If it does, a human agent who is sending a message of type IR and who has in his initial knowledge InitK('Human',IR,R) and has to send it to agent R. That restrictions is a little more elaborate than the ones we considered, furthermore it supposes an initial knowledge of the human (that is not forgotten).

```
restriction noTellEx:
''All IR R I m #i #s #s2. Rule4('Human','noTellEx',I,IR)@s & InitK('Human',IR,R) @s2 &
Send('Human',I,m) @i ==> To(R) @i''
```

Fig. 13: The rule noTellEx defined in Tamarin

Third rule The third rule defined in Tamarin can be found in Figure 14. For the rule to work, an action Rule3('Human','noGet',I) needs to appear on the traces of interest. If it does, a human agent cannot receive any message of type I.

```
restriction noGet:
''All I x #s #i. Rule3('Human','noGet',I) @s & Receive('Human',I,x) @i ==> F''
```

Fig. 14: The rule noGet defined in Tamarin

Fourth rule The forth rule defined in Tamarin can be found in Figure 15. For the rule to work, an action Rule3('Human','ICompare',I) needs to appear on the traces of interest. If it does, a human agent cannot receive any message of type I unless he has it in his initial knowledge.

```
restriction ICompare:
'' All I y #s #i. Rule3('Human','ICompare',I)@s & Receive('Human',I,y) @i
==> Ex #j. InitK('Human',I,y) @j''
```

Fig. 15: The rule ICompare defined in Tamarin

Overall, the rules defined here are similar to the ones we used but they do not allow to compare a message to send or receive with a message previously sent or received. That is why our restrictions are a little more effective to restore the correctness of protocols.

Time to compute	Number of strategies on which Tamarin computed a result	Number of strategies on which Tamarin took too much time	Number of strategy whose result was deduced
1473.5	12	5	7
3386.6	13	12	39
6313.7	22	27	77
8158.4	32	30	79
8506.8	30	26	56

Fig. 16: The result got on the strategies with the computational time (in seconds) it required to find the correct one. This was done on the security protocol MPAuth_VC with different random seed. In every cases, the set of restrictions is the same, there are 255 of them. The timeout is equal to $240.0s$.

D Effectiveness of the exploration of the set of strategies

In Figure 16 is displayed several computational times necessary to find an appropriate strategy. The only thing that changes between all these cases is the random seed. The second column shows the number of strategies on which Tamarin was launched and had the time to compute a result. The third one shows the number of strategies on which Tamarin was launched but did not have the time to compute a result. Finally, the fourth column shows the number of strategies on which Tamarin was not launched but we could deduce with the observations of section 5 that they could not restore the correctness of the protocol.

First, we can see that the computational time to find a good strategy seems to highly depend on the number of strategies on which Tamarin was launched.

Moreover, it seems that around half of the strategies on which it is launched, Tamarin does not have the time to conclude. That could be an indication that the timeout could be set up higher than what it was when the computation was done. However, that would not guarantee that more results are computed.

To evaluate the effectiveness of our exploration of the set of strategies, we can look at the second and the fourth columns. If we do not consider the first case, for which the correct strategy was found very quickly, we can see that the number of results deduced is between 2 and 3 times bigger than the number of strategies on which we have a result given by Tamarin. This tends to show that from a few result, quite a lot information can be deduced. However, we would need to compare this result with other protocols to see if this is a specificity of this protocol or if this is mostly due to the structure of the set of strategies we consider.