Parameterized Verification of Timed Security Protocols with Clock drift

Li Li, **Jun Sun** and Jin Song Dong

Motivation

"Since clock synchronization is so important in the security of the Kerberos protocol, if clocks are not synchronized within a reasonable window Kerberos will report fatal errors and refuse to function."

It is advisable to set Maximum tolerance for computer clock synchronization to a value of 5 minutes.

What kind of clock drifts are safe?
How do we formally answer such questions?

History

“Verifying Parameterized Timed Security Protocols” (FM 2015)

Finding: a timing attack in Kerberos V specification.

We are responsible for answering the questions!

Verifying Parameterized Timed Security Protocols

Li Li[1], Jun Sun[2], Yang Liu[3], and Jin Song Dong[1]

[1] National University of Singapore
[2] Singapore University of Technology and Design
[3] Nanyang Technological University

Abstract. Quantitative timing is often explicitly used in systems for better security, e.g., the credentials for automatic website logon often has limited lifetime. Verifying timing relevant security protocols in these systems is very challenging as timing adds another dimension of complexity compared with the untimed protocol verification. In our previous work, we proposed an approach to check the correctness of the timed authentication in security protocols with fixed timing constraints. However, a more difficult question persists, i.e., given a particular protocol design, whether the protocol has security flaws in its design or it can be configured secure with proper parameter values? In this work, we answer this question by proposing a parameterized verification framework, where the quantitative parameters in the protocols can be intuitively specified as well as automatically analyzed. Given a security protocol, our verification algorithm either produces the secure constraints of the parameters, or constructs an attack that works for any parameter values. The correctness of our algorithm is formally proved. We implement our method into a tool called PTAuth and evaluate it with several security protocols. Using PTAuth, we have successfully found a timing attack in Kerberos V which is unreported before.

1 Introduction

Time could be a powerful tool in designing security protocols. For instance, distance bounding protocols rely heavily on time; session keys with limited lifetime are extensively used in practice to achieve better security. However, designing timed security protocols is more challenging than designing untimed ones because timing adds a range of attacking surface, e.g., the adversary might be able to extend the session key without proper authorization. Hence, it is important to have a formal verification framework to analyze the timed security protocols. In our previous work [20], we developed a verification algorithm to analyze whether a given protocol with fixed timing constraints is secure or not. In this work, we answer a more difficult question, i.e., given a security protocol with configurable parameters for the timing constraints, are there any parameters which could guarantee security and what are they? Having an approach to answer the question is useful in a number of ways. Firstly, it can analyze, at once, all instances

Research Questions

How do we model timed security protocols?

How do we model clock drifts?

How do we verify the models?

A Running Example

Corrected Wide Mouthed Frog (WMF)

- a key exchange protocol
- verified to be secure assuming clocks are perfectly synchronized

Server

Alice

Bob

Corrected WMF

1. send <t_a, B, k, tag1>
encrypted using key(A)

2. receive at t_s
check t_s - t_a <= p

3. send <t_s, A, k, tag2>
encrypted using key(B)

4. receive at t_b
check t_b - t_s <= p
accepts session key k

Modeling Corrected WMF

Timed Applied π-Calculus

1. send <t_a, B, k, tag1>
encrypted using key(A)

$$P_a \triangleq in(r).\nu k.\mu t_a : c_a.init(A, r, k)@t_a.\overline{out}(\langle A, enc_s(\langle t_a, r, k, tag_1\rangle, key(A))\rangle).0$$

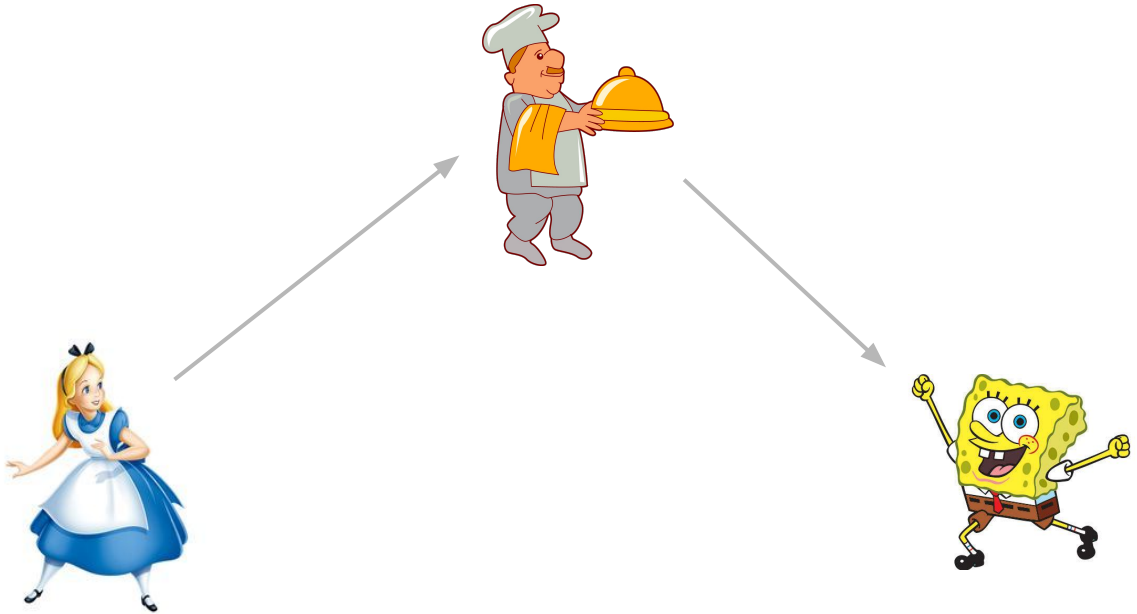
Modeling Corrected WMF

Timed Applied π-Calculus

4. receive at t_b
check t_b - t_s <= p
accepts session key k

$$P_b \triangleq c(x).\mu t_b : c_b.let\ \langle t_s, =A, k, =tag_2\rangle = dec_s(x, key(B))\ then$$
$$if\ t_b - t_s \leq p_m\ then\ check\ k\ in\ db\ as\ unique\ then\ accept(A, B, k)@t_b.0$$

Modeling Corrected WMF



$$P \triangleq (!P_r)|(\bar{!}P_a)|(!P_s)|(!P_b)|(!P_p)$$

Timed Logic Rules

$$[\ G\ ]\ e_1, e_2, \ldots, e_n\ -[\ B\ ]\rightarrowtail\ e$$

G: an untimed guard condition; e: an event; B: a timed constraint

Rules from the protocol model

Rules modeling the attacker

Model Rules

$$know(\mathbb{t}_b, \mathbb{t}_b), know(enc_s(\langle \mathbb{t}_s, A[], k, tag_2[] \rangle, key(B[])), \mathbb{t}_1)$$
$$-[\ \mathbb{t}_1 \leq \mathbb{t}_b \wedge \mathbb{t}_b - \mathbb{t}_s \leq \S p_m\ ]\rightarrowtail accept([n_b], \langle A[], B[], k \rangle, \mathbb{t}_b)$$

Assume no clock drift now

4. receive at t_b
check t_b - t_s <= p
accepts session key k

Attacker Model

Delov-Yao Attacker Model, e.g.

$$know(m, \mathbb{t}_1), know(k, \mathbb{t}_2) \dashv [\ \mathbb{t}_1 \leq \mathbb{t} \wedge \mathbb{t}_2 \leq \mathbb{t}\ ] \vdash know(enc_s(m, k), \mathbb{t})$$

More than Delov-Yao, e.g.

$$know(RC4(m, k), t_1)\rangle \dashv [\ t - t_1 > \S d\ ] \vdash know(k, t)$$

Modeling Clock Drift

VR (Variable Rate):

Different clocks have different clock rates and there is a maximum bound on the drift

SR (Same Rate):

Different clocks share the clock rate but have different readings

Clock Drift: VR

$$know(\mathbb{t}_b, \mathbb{t}_b), know(enc_s(\langle \mathbb{t}_s, A[], k, tag_2[] \rangle, key(B[])), \mathbb{t}_1)$$
$$\dashv\!\lfloor\, \mathbb{t}_1 \leq \mathbb{t}_b \wedge \mathbb{t}_b - \mathbb{t}_s \leq \S p_m \,\rfloor\!\vdash\, accept([n_b], \langle A[], B[], k \rangle, \mathbb{t}_b)$$

$$know(\mathbb{t}_b, \mathbb{t}'_b), know(enc_s(\langle \mathbb{t}_s, A[], k, tag_2[] \rangle, key(B[])), \mathbb{t}_1)$$
$$\dashv\!\lfloor\, \mathbb{t}_1 \leq \mathbb{t}'_b \wedge \mathbb{t}_b - \mathbb{t}_s \leq \S p_m \wedge |\mathbb{t}'_b - \mathbb{t}_b| \leq \S p_b \,\rfloor\!\vdash\, accept([n_b], \langle A[], B[], k \rangle, \mathbb{t}'_b)$$

Clock Drift: SR

$$know(\mathbb{t}_b, \mathbb{t}_b), know(enc_s(\langle \mathbb{t}_s, A[], k, tag_2[] \rangle, key(B[])), \mathbb{t}_1)$$
$$\dashv\!\lfloor\ \mathbb{t}_1 \leq \mathbb{t}_b \wedge \mathbb{t}_b - \mathbb{t}_s \leq \S p_m\ \rfloor\!\vdash\ accept([n_b], \langle A[], B[], k \rangle, \mathbb{t}_b)$$

$$know(\mathbb{t}_b, \mathbb{t}'_b), know(enc_s(\langle \mathbb{t}_s, A[], k, tag_2[] \rangle, key(B[])), \mathbb{t}_1)$$
$$\dashv\!\lfloor\ \mathbb{t}_1 \leq \mathbb{t}'_b \wedge \mathbb{t}_b - \mathbb{t}_s \leq \S p_m \wedge \mathbb{t}_b - \mathbb{t}'_b = \S d_b\ \rfloor\!\vdash\ accept([n_b], \langle A[], B[], k \rangle, \mathbb{t}'_b)$$

Research Questions

How do we model timed security protocols?

How do we model clock drifts?

How do we verify the models?

Verification: Property

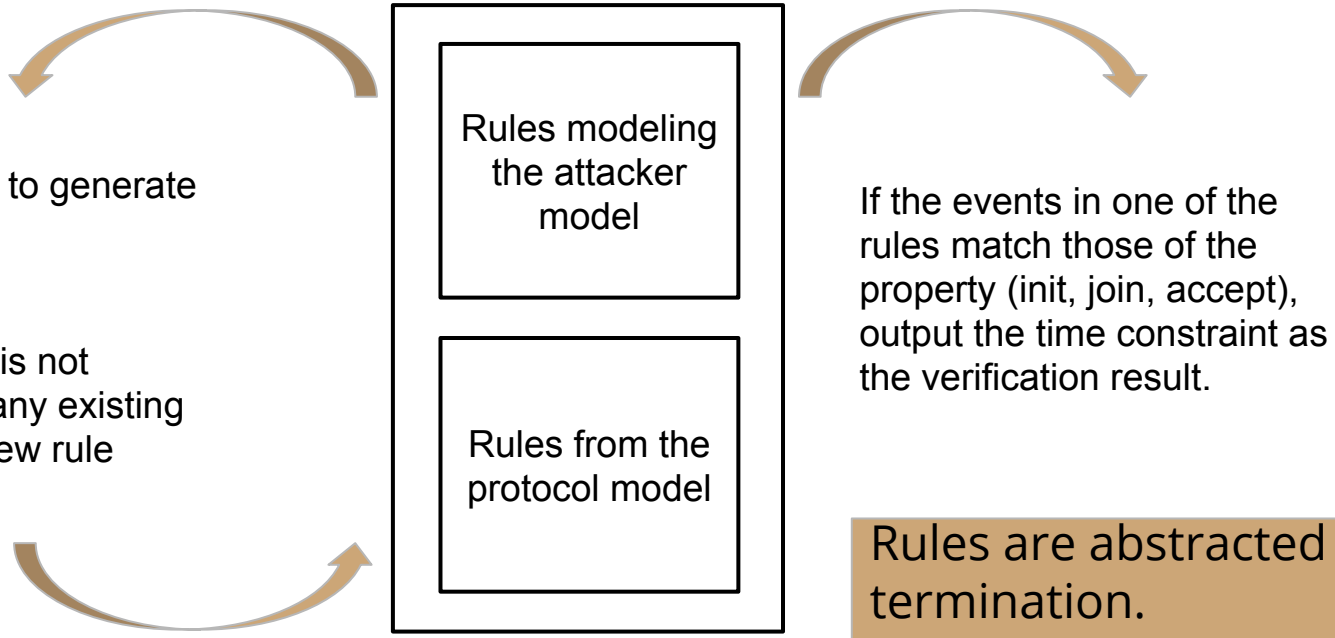
Non-injective timed authentication

> *For every acceptance of the protocol responder, the protocol initiator indeed initiates the protocol the protocol and protocol partners indeed join in the protocol, agreeing on the protocol arguments and timing requirements.*

$$accept(i,r,k)@t_r \leftarrow\!\!\lfloor\ t_s - t_i \leq \S p_m$$
$$\wedge\, t_r - t_s \leq \S p_m \rfloor\!\!\rightarrow\ init(i,r,k)@t_i, join(i,r,k)@t_s$$

Another rule.

Verification Algorithm



Rule Composition

$$know(m, \mathbb{t}_1), know(k, \mathbb{t}_2) \dashv\!\!\lfloor\ \mathbb{t}_1 \leq \mathbb{t} \wedge \mathbb{t}_2 \leq \mathbb{t}\ \rfloor\!\!\mapsto know(enc_s(m, k), \mathbb{t})$$

+

$$know(\mathbb{t}_b, \mathbb{t}_b), know(enc_s(\langle \mathbb{t}_s, A[], k, tag_2[] \rangle, key(B[])), \mathbb{t}_1)$$
$$\dashv\!\!\lfloor\ \mathbb{t}_1 \leq \mathbb{t}_b \wedge \mathbb{t}_b - \mathbb{t}_s \leq \S p_m\ \rfloor\!\!\mapsto accept([n_b], \langle A[], B[], k \rangle, \mathbb{t}_b)$$

||

$$know(\mathbb{t}_b, \mathbb{t}_b), know(\langle \mathbb{t}_s, A[], k, tag_2[] \rangle, \mathbb{t}_1), know(key(B[]), \mathbb{t}_2)$$
$$\dashv\!\!\lfloor\ \mathbb{t}_1 \leq \mathbb{t}_b \wedge \mathbb{t}_2 \leq \mathbb{t}_b \wedge \mathbb{t}_b - \mathbb{t}_s \leq \S p_m\ \rfloor\!\!\mapsto accept([n_b], \langle A[], B[], k \rangle, \mathbb{t}_b)$$

Evaluation

Protocol	$\sharp\mathcal{R}$	No Clock Drift		Shared Clock Rate		Variable Clock Rate	
		Result	Time	Result	Time	Result	Time
Corrected WMF [7,18,16]	80	Secure	47.51ms	Threat	112.75ms	Attack	150.09ms
TESLA [22,21]	343	Secure	3.17s	Threat	3.55s	Threat	4.37s
Auth Range [6,8]	53	Secure	38.58ms	Secure	60.73ms	Attack	46.47ms
CCITT X.509 (1c) [3]	135	Secure	162.69ms	Secure	231.86ms	Secure	224.00ms
CCITT X.509 (3) BAN [7]	198	Secure	791.00ms	Secure	1058.05ms	Secure	969.97ms
NS PK Time [20,17,10]	173	Secure	170.00ms	Threat	205.93ms	Threat	353.20ms

Secure: some trivial time constraint has to be satisfied
Threat: some nontrivial constraint has to be satisfied
Attack: there is always an attack

Case Study: TELSA

Designed with clock drifts

No clock drift or Shared Clock Rates:

Verification Result: 2*network latency < interval

Variable Clocks:

Verification Result: drift_s + drift_r <= interval

Conclusion

We have developed a tool to verify security protocols with clock drifts.

This line of work is based on ProVerif.

Details: “Automated Verification of Time Security Protocols with Clock Drift”, FM 2016.

Ongoing Work

"Since clock synchronization is so important in the security of the Kerberos protocol, if clocks are not synchronized within a reasonable window Kerberos will report fatal errors and refuse to function."

It is advisable to set Maximum tolerance for computer clock synchronization to a value of 5 minutes.

Unfortunately, the current implementation is not efficient enough to verify Kerberos V once clock drift is considered.