Formal Verification of Security Protocols

Vaishnavi Sundararajan
Chennai Mathematical Institute

SRM, Chennai 13 March, 2018

Software Testing

- Programming is invariably buggy.
- How does one find bugs before actual use?
- Throw test inputs at program, see if correct.
- Try arcane corner cases, do sandbox testing.
- Ultimately, hope for the best!

Software Testing (Contd.)

- Enough to try many corner case inputs?
- What about programs for online use? Interaction?
- How much is enough? When does one stop?
- What about software manipulating money? Or passwords?

"Testing shows the presence, not the absence of bugs."

- Edsger W. Dijkstra

It's A Bird, It's A Plane...

- It's formal verification!
- Allows programs to be proved correct, also finds bugs.
- Especially important for programs handling sensitive data.

Formal Verification: Overview

- Theorem provers have been around since 1950s.
- 1994: Intel discovers floating-point-division bug. Starts formally verifying all chips.
- Formal verification sort of catches on in industry.

Formal Verification: Overview

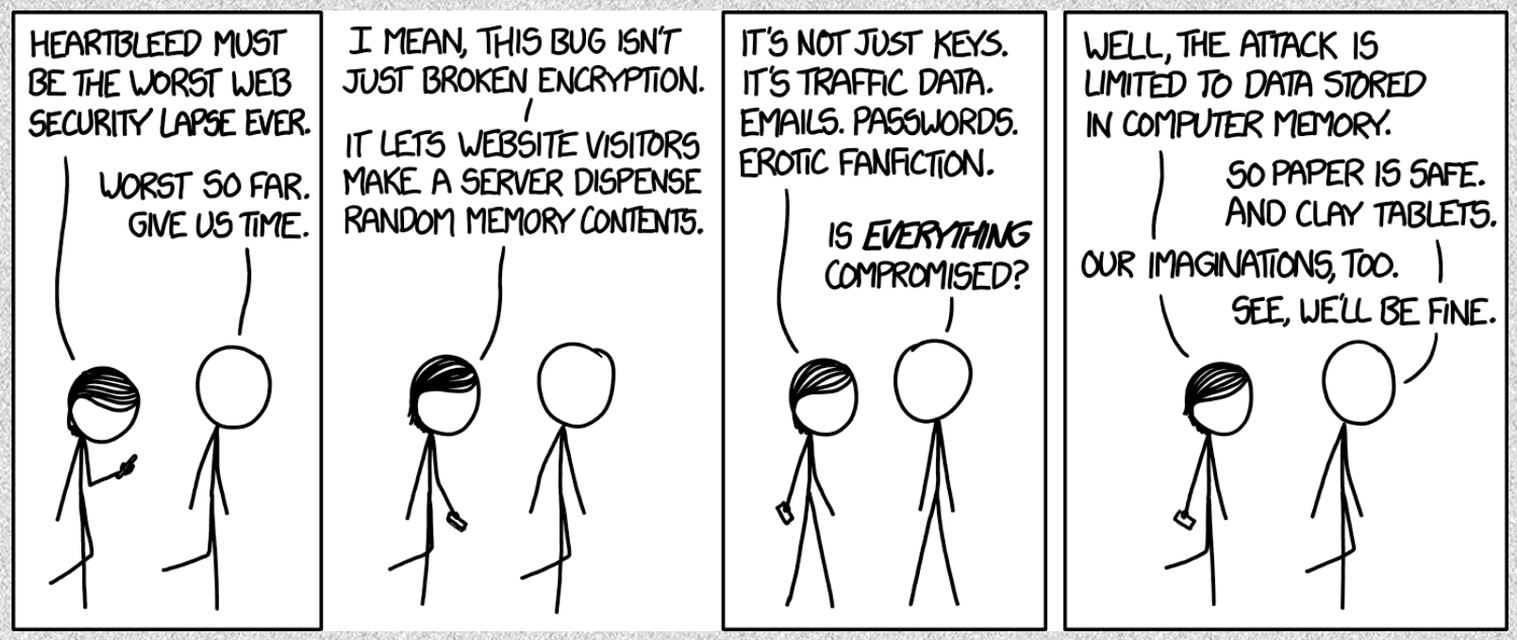
- Two ways: Model checking, and deductive inference.
- Model checking: Create symbolic model that captures key behaviour and ignores extraneous details. Define properties in this model and verify.
- Deductive inference: Specify pre-execution and post-execution statements in some logical language, where latter can be proved from former in a 'correct' program.

Security

- Security inside a system: Input sanitisation, memory restrictions, appropriate buffer allocations…
- Security across systems: Firewalls, IP filtering, secure communication protocols…

Formal Methods For Security

- Bugs continue to haunt security protocols.
- OpenSSL suffered a bug called Heartbleed.
- Single-Sign-On was found to be flawed by Duo Labs.



Formal Methods For Security

- Focus on symbolic model checking in this talk.
- Idealised model of communication and adversary
- Adversary controls network, can see all communication
- Cryptography assumed perfect — PKI, hashing etc.
- Formalise security properties and verify!

Formal Methods For Security

- Abstract communications of the form $A \rightarrow B : t$
- A sends to B a term t (constant, fresh random number, pair, encrypted term etc)
- Encryption represented as function $enc(t, k)$
- Shorthand notation $\{t\}_k$

A Simple Example

- *A* wants to check if *B* is online.
- Simple protocol to achieve this:

$$A \longrightarrow B : \{x\}_{pk(B)}$$

$$B \longrightarrow A : \{x\}_{pk(A)}$$

- At the end of this protocol, can *A* be sure that *B* is online?
- Now that x is known to *A* and *B*, can it be used as a shared secret between them (perhaps to authenticate further communication)?

A Simple Example (Onlineness)

- At the end of this, can *A* be sure that *B* is online?


DEPENDS ON WHAT
THE MEANING OF THE WORD "IS" IS

A Simple Example (Onlineness)

- What if *A* reuses an *x* she used earlier to talk to *C*?
- *I* can replay C's response from then.
- *B* might not be online.
- But *A* thinks he is!

A Simple Example (Onlineness)

- *I* can replay an earlier message!

$$A \rightarrow C : \{x\}_{pk(C)}$$

$$C \rightarrow A : \{x\}_{pk(A)}$$

LATER

$$A \rightarrow B : \{x\}_{pk(B)}$$

$$(I)\ B \rightarrow A : \{x\}_{pk(A)}$$

A Simple Example (Onlineness)

- Ensure that x is "fresh" (randomly picked!) for every new session.

A Simple Example (Onlineness)

- Ensure that x is "fresh" (randomly picked!) for every new session.

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

A Simple Example (Authentication)

- If x is known to only A and B, can it be used as a shared secret between them (perhaps to authenticate further communication)?

- Again…


DEPENDS ON WHAT
THE MEANING OF THE WORD "IS" IS

A Simple Example (Authentication)

- Is it even the case that x is always known only to A and B?
- NOPE.
- I can execute a man-in-the-middle attack!

$$A \longrightarrow (I)\ B : \{x\}_{pk(B)}$$

$$I \longrightarrow B : \{x\}_{pk(B)}$$

$$B \longrightarrow I : \{x\}_{pk(I)}$$

$$(I)\ B \longrightarrow A : \{x\}_{pk(A)}$$

A Simple Example (Authentication)

- We would like x to be secret to A and B.
- Easy fix: Include sender's identity inside encryption.

$$A \rightarrow B : \{A, x\}_{pk(B)}$$

$$B \rightarrow A : \{x\}_{pk(A)}$$

- Sounds fine, but how do we know this fixed the issue?

Proving Correctness

- *A* sends out $\{A, x\}_{pk(B)}$ intended for *B*. Can we show that *I* never gets to learn *x*?

- Consider the various ways in which it is possible for *I* to have learnt *x*.

- If *I* never gets to learn *x*, each of these ways should result in some sort of contradiction.

Dolev-Yao Model

- Abstracts away from the 0/1 bit-world to a symbolic model of communication.
- Messages are abstract terms rather than bitstrings.
- Encryption, hashing etc. abstract functions on terms.
- Crypto assumed to be perfect, no cryptanalysis!

Dolev-Yao Model: Intruder

Intruder *I* cannot break encryption, but can

- see any message
- block any message
- redirect any message
- generate messages — according to set rules!
- send messages in someone else's name
- start new session of the protocol

Dolev-Yao Model: Actions

- Each communication is separated out into two actions: a send action and a 'corresponding' receive action.

- Every term sent out is interpreted as being received by *I*, and each received term is assumed to be coming from *I*. (Ties in well with our intuition of *I* being the network!)

$$t := m \mid (t_1, t_2) \mid \{t\}_k$$

$$\frac{}{X \vdash t}\ ax\ \ (t \in X)$$

$$\frac{X \vdash (t_0, t_1)}{X \vdash t_i}\ split_i\ \ (i = 0, 1) \qquad \frac{X \vdash t_0 \quad X \vdash t_1}{X \vdash (t_0, t_1)}\ pair$$

$$\frac{X \vdash \{t\}_k \quad X \vdash inv(k)}{X \vdash t}\ dec \qquad \frac{X \vdash t \quad X \vdash k}{X \vdash \{t\}_k}\ enc$$

Term derivation system

Dolev-Yao Model: Attacks (Again)

Recall this attack.

$$A \rightarrow (I)\ B : \{x\}_{pk(B)}$$

$$I \rightarrow B : \{x\}_{pk(B)}$$

$$B \rightarrow I : \{x\}_{pk(I)}$$

$$(I)\ B \rightarrow A : \{x\}_{pk(A)}$$

How do we formally show that I learns the value of x here?

Dolev-Yao Model: Attacks (Again)

Need to analyse what terms get added to I's database.

$$A \rightarrow (I)\ B : \{x\}_{pk(B)}$$

$$I \rightarrow B : \{x\}_{pk(B)}$$

$$B \rightarrow I : \{x\}_{pk(I)}$$

$$(I)\ B \rightarrow A : \{x\}_{pk(A)}$$

Separate out each communication into the constituent send and receive actions, and see what terms get added to I's database.

Dolev-Yao Model: Attacks (Again)

$$A \rightarrow (I)\ B : \{x\}_{pk(B)}$$

$$I \rightarrow B : \{x\}_{pk(B)}$$

$$B \rightarrow I : \{x\}_{pk(I)}$$

$$(I)\ B \rightarrow A : \{x\}_{pk(A)}$$

Dolev-Yao Model: Attacks (Again)

$A \rightarrow (I)\ B : \{x\}_{pk(B)}$ $\quad +A: \{x\}_{pk(B)}$

$I \rightarrow B : \{x\}_{pk(B)}$ $\quad -B: \{x\}_{pk(B)}$

$B \rightarrow I : \{x\}_{pk(I)}$ $\quad +B: \{x\}_{pk(I)}$

$(I)\ B \rightarrow A : \{x\}_{pk(A)}$ $\quad -A: \{x\}_{pk(A)}$

Dolev-Yao Model

- Names *A*, *B* etc. denote 'roles', not specific agents.
- Each name instantiated to a concrete value in a run.
- Each run has multiple sessions with complex interactions; need to keep track of all instantiations.
- Run mechanism modelled in many different ways — automata, processes etc.

Security Properties

- Secrecy: Intruder should not learn a designated secret
- Correspondence: A happens only if B happened earlier.
- Can verify by verifying each individual run.

Security Properties

- What about properties spanning runs?

- Anonymity: No link between voter and vote.

- Maybe combining info across runs violates anonymity!

- Intuitively harder to verify than examining each run.

Good News, Bad News

- Bad news first.
- General verification problem is undecidable. Boo.

Good News, Bad News

- Bad news first.
- General verification problem is undecidable. Boo.
- Good news now!
- Solvable for restricted (but meaningful) classes: finitely many sessions, boundedly many names etc.

Verification In Real Life

- Attack on anonymity for Helios e-voting protocol.
- Whatsapp/FB messenger's underlying protocol recently analyzed.

A Formal Security Analysis of the Signal Messaging Protocol

Katriel Cohn-Gordon[1], Cas Cremers[1], Benjamin Dowling[2], Luke Garratt[1], and Douglas Stebila[3]

[1]University of Oxford, Oxford, UK. first.last@cs.ox.ac.uk
[2]Queensland University of Technology, Brisbane, Australia. b1.dowling@qut.edu.au
[3]McMaster University, Hamilton, Ontario, Canada. stebilad@mcmaster.ca

Abstract

Signal is a new security protocol that provides end-to-end encryption for instant messaging. It has recently been adopted by WhatsApp, Facebook Messenger and Google Allo among many others; the first two of these have at least 1 billion active users. Signal includes several uncommon security properties (such as "future secrecy" or "post-compromise security"), enabled by a novel technique called *ratcheting* in which session keys are updated with every message sent. Despite its importance and novelty, there has been little to no academic analysis of the Signal protocol.

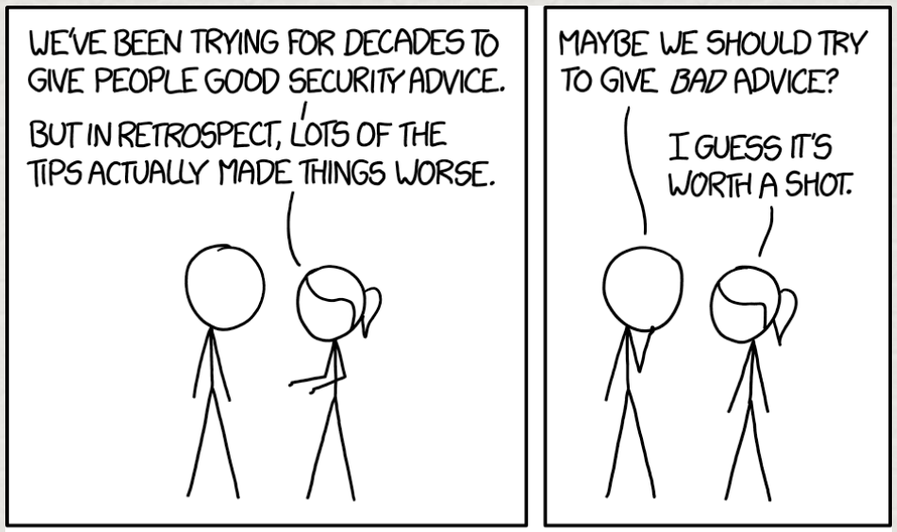
We conduct the first security analysis of Signal's Key Agreement and Double Ratchet as a multi-stage key exchange protocol. We extract from the implementation a formal description of the abstract protocol, define a security model which can capture the "ratcheting" key update structure, and prove the security of Signal's core in our model. Our presentation and results can serve as a starting point for other analyses of this widely adopted protocol.

Verification In Real Life (Contd.)

- Lots of tools available to automate verification.
- Security-centric: Scyther, Statverif, Tamarin…
- Automated theorem provers: Isabelle, Coq…
- Hotbed of research in various labs like IBM, MSR, Google etc.

Conclusion

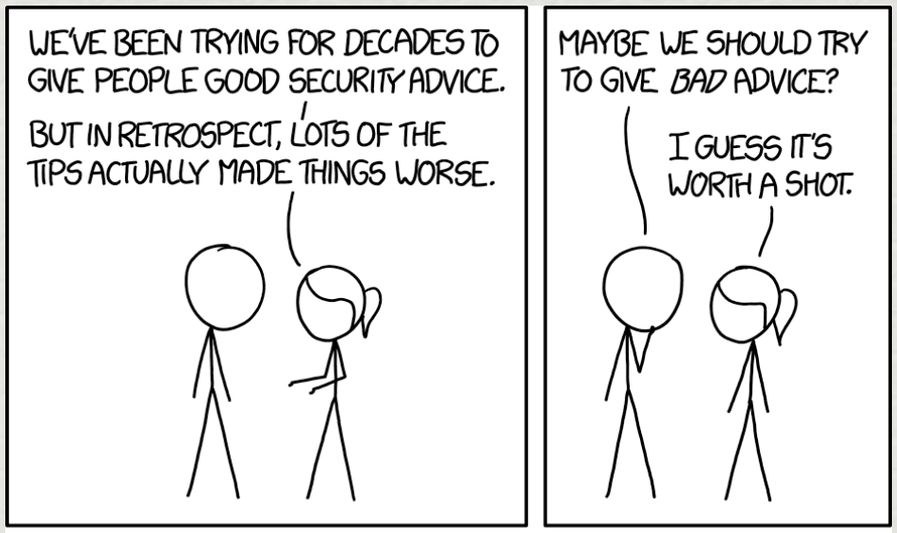
- Formal verification of protocols is important.
- Symbolic models are used to abstractly capture behaviour.
- Verification of these models presents many interesting questions about derivability, automation etc.
- Great area to come do cutting-edge research in!



SECURITY TIPS

(PRINT OUT THIS LIST AND KEEP IT IN YOUR BANK SAFE DEPOSIT BOX.)

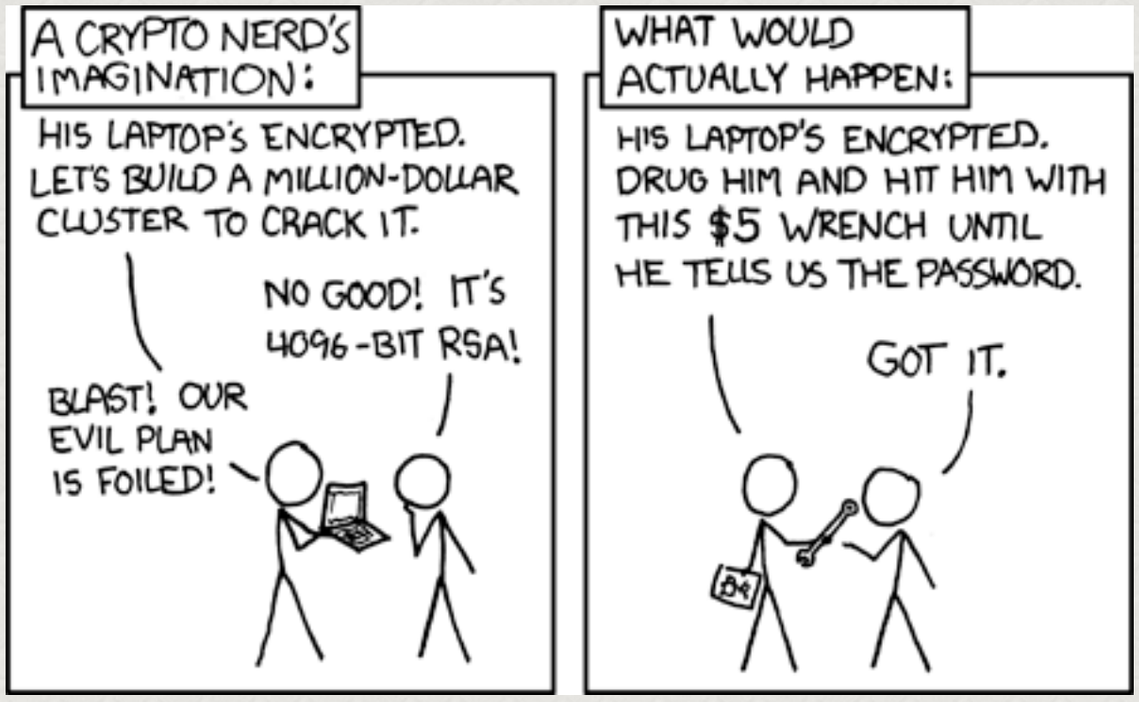
- DON'T CLICK LINKS TO WEBSITES
- USE PRIME NUMBERS IN YOUR PASSWORD
- CHANGE YOUR PASSWORD MANAGER MONTHLY
- HOLD YOUR BREATH WHILE CROSSING THE BORDER
- INSTALL A SECURE FONT
- USE A 2-FACTOR SMOKE DETECTOR
- CHANGE YOUR MAIDEN NAME REGULARLY
- PUT STRANGE USB DRIVES IN A BAG OF RICE OVERNIGHT
- USE SPECIAL CHARACTERS LIKE & AND %
- ONLY READ CONTENT PUBLISHED THROUGH TOR.COM
- USE A BURNER'S PHONE
- GET AN SSL CERTIFICATE AND STORE IT IN A SAFE PLACE
- IF A BORDER GUARD ASKS TO EXAMINE YOUR LAPTOP, YOU HAVE A LEGAL RIGHT TO CHALLENGE THEM TO A CHESS GAME FOR YOUR SOUL.



SECURITY TIPS

(PRINT OUT THIS LIST AND KEEP IT IN YOUR BANK SAFE DEPOSIT BOX.)

- DON'T CLICK LINKS TO WEBSITES
- USE PRIME NUMBERS IN YOUR PASSWORD
- CHANGE YOUR PASSWORD MANAGER MONTHLY
- HOLD YOUR BREATH WHILE CROSSING THE BORDER
- INSTALL A SECURE FONT
- USE A 2-FACTOR SMOKE DETECTOR
- CHANGE YOUR MAIDEN NAME REGULARLY
- PUT STRANGE USB DRIVES IN A BAG OF RICE OVERNIGHT
- USE SPECIAL CHARACTERS LIKE & AND %
- ONLY READ CONTENT PUBLISHED THROUGH TOR.COM
- USE A BURNER'S PHONE
- GET AN SSL CERTIFICATE AND STORE IT IN A SAFE PLACE
- IF A BORDER GUARD ASKS TO EXAMINE YOUR LAPTOP, YOU HAVE A LEGAL RIGHT TO CHALLENGE THEM TO A CHESS GAME FOR YOUR SOUL.



Thank You!

vaishnavi@cmi.ac.in

http://www.cmi.ac.in/~vaishnavi/