CS3235 - Computer Security Final lecture - Machine architecture 2

Hugh Anderson

National University of Singapore
School of Computing

November, 2018

Hiding places...

Outline

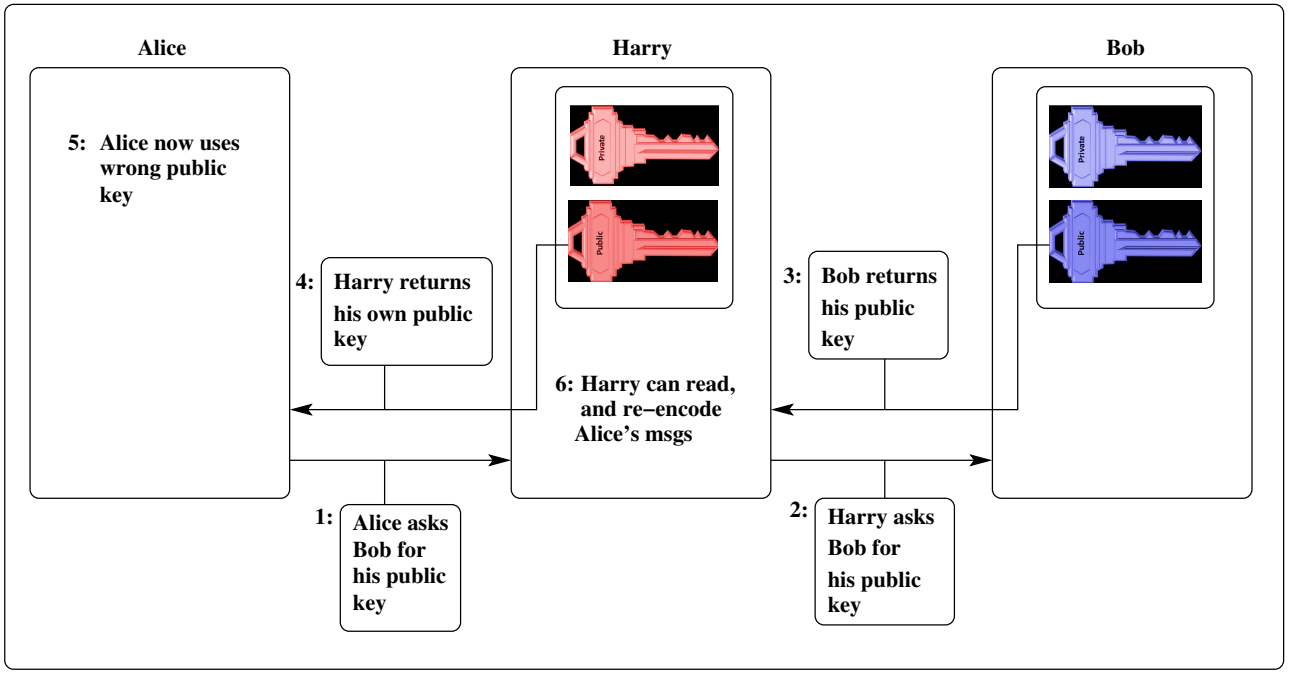
Outline

Outline

Outline

Case Study 1: PKI

Motivation for PKI...



Summary:

The overall mechanism provides certificates that contain public keys, bound to identities (web sites for example). Certifying authorities provide some level of trust, providing certificates.

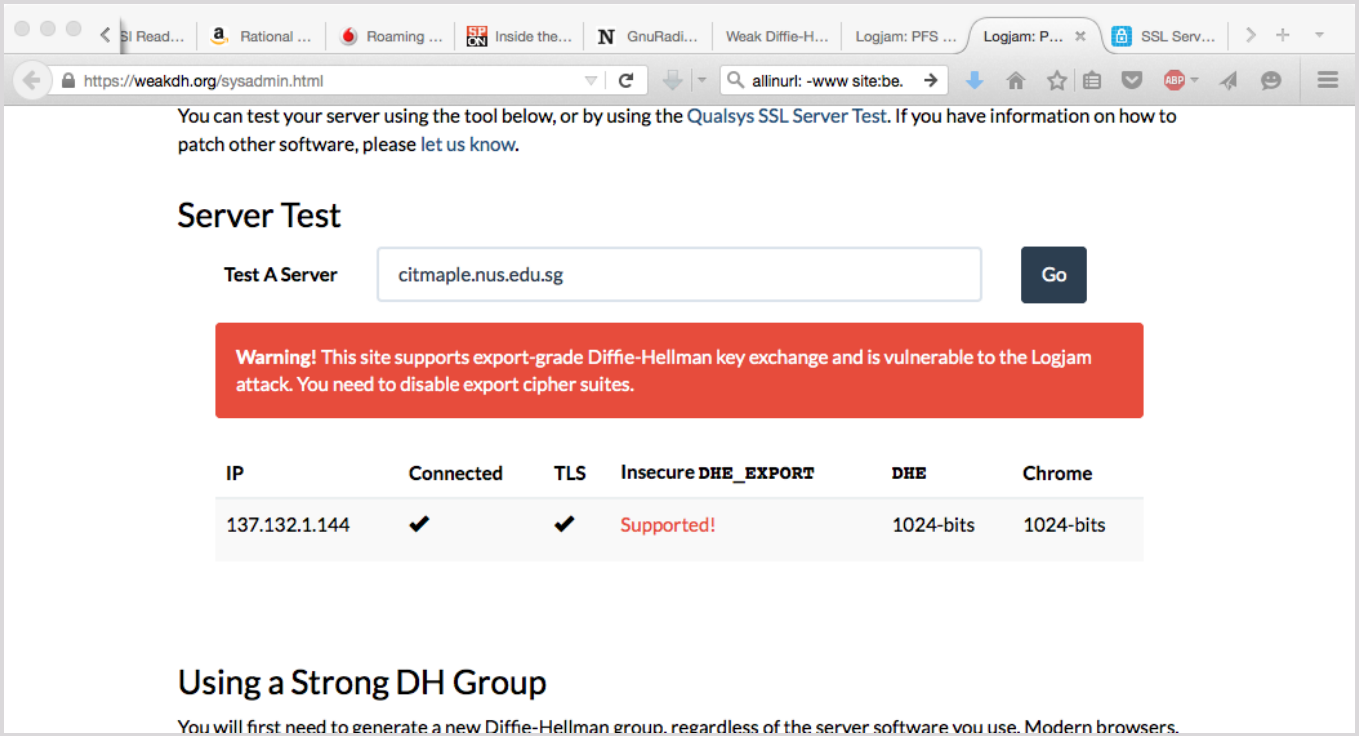
Case study 1: Monitoring - should we care?

June 2013: Snowden releases NSA documents...

... and in them we discover that NSA has been routinely listening in on skype calls, and intercepting IPsec and https:// conversations.

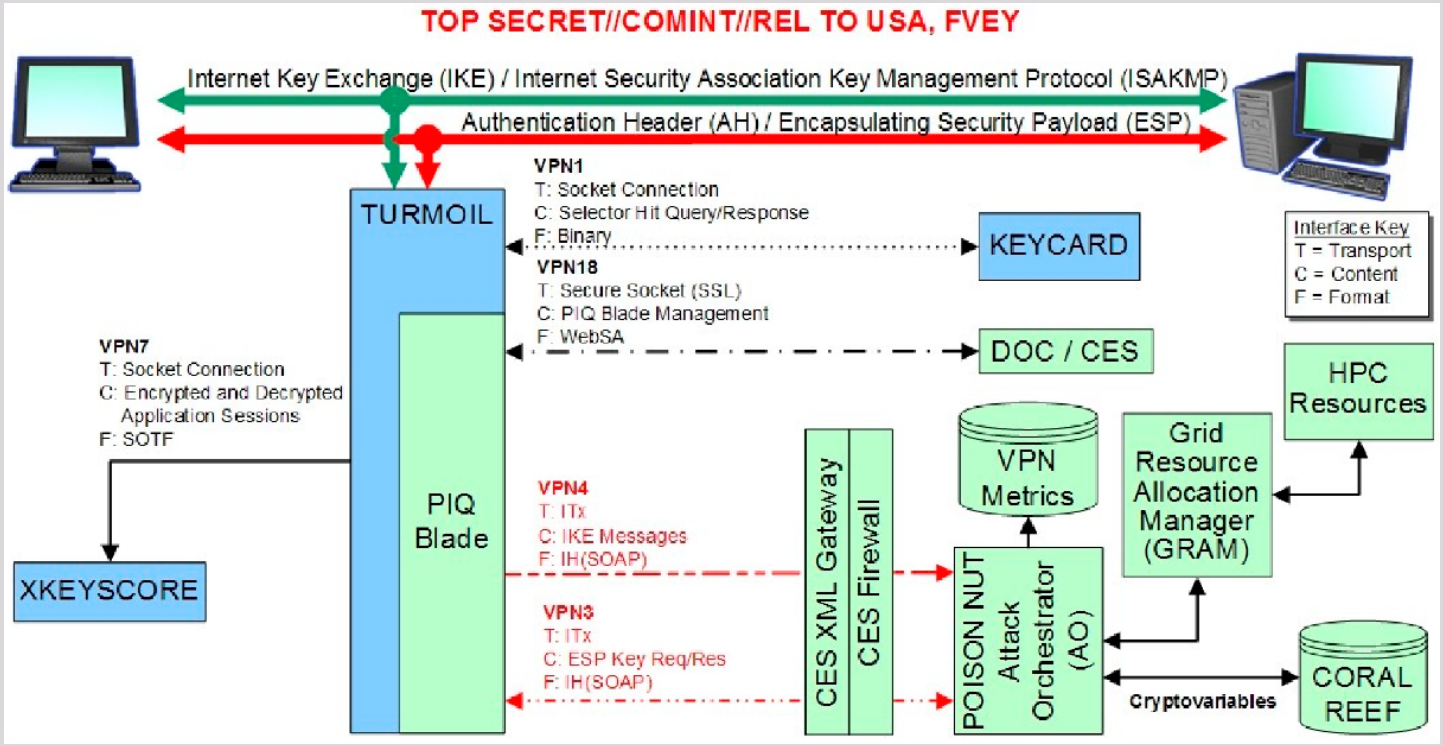
My colleague Martin suggests that, in this day and age, most people couldn't care less. Maybe he is right. But in any case, how can it be done?

October 2015: Plausible explanation... https://weakdh.org/



Case study 1: Large scale monitoring of IPSec

Architecture of an NSA system from leaked document



Notice the diagram is clearly indicating the IKE (Internet Key Exchange), and the use of IPsec - the AH and ESP packets.
There is so much supporting material, that it is clear that this is not FUD, but is a real mechanism, in use since at least 2009.
It relies on the use of the fixed large primes used in Diffie-Hellman key exchange software.

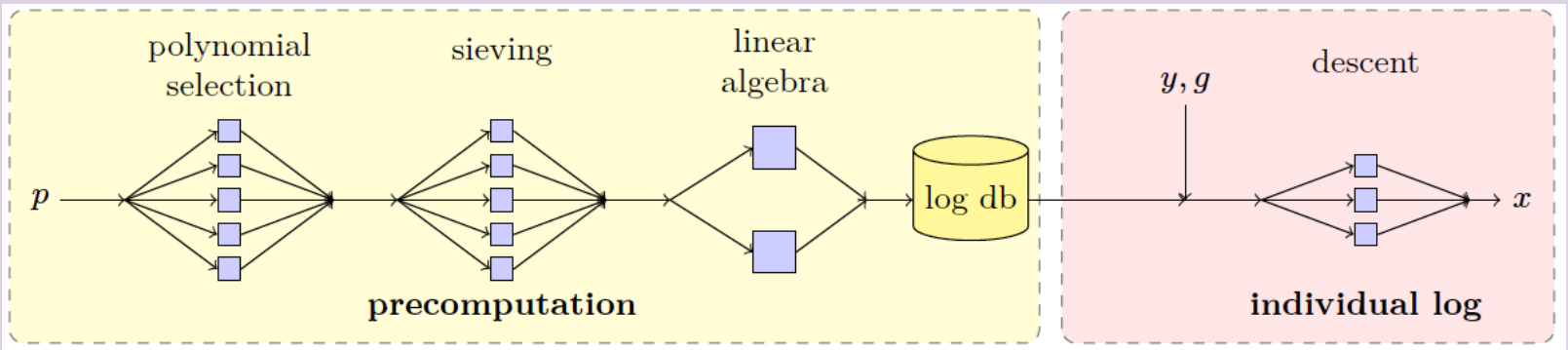
Case study 1: Large scale monitoring of IPSec

What are the steps?

- During initial setup of the link, downgrade the Diffie-Hellman key exchange to one using smaller (EXPORT-GRADE) bitsize primes.
- Send IKE (Internet Key Exchange) values to supercomputer, which returns actual keys within 15 minutes.
  - October 2015 paper succeeds in 70 seconds: logjam - a mitm downgrade to 512 bit primes.

Solved $y = g^x \bmod p$ discrete log problem? ... No.

Common (NFS) algorithm can use pre-computation, for a specific prime, of 3 out of the 4 steps:



Case study 1: Large scale monitoring of IPSec

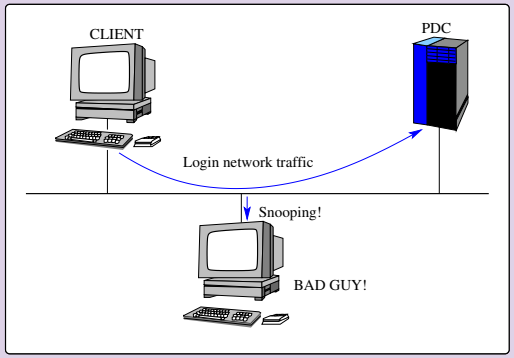
Solutions? (From the paper)

- Browser - check you have the updates which stop *you* being vulnerable
- Server...
  - Disable Export Cipher Suites. There is little downside in disabling them.
  - Deploy (Ephemeral) Elliptic-Curve Diffie-Hellman (ECDHE). No known feasible cryptanalytic attacks
  - Use a Strong, Diffie Hellman Group. Use 2048-bit or stronger Diffie-Hellman groups with "safe" primes.

Outline

Case study 2: Challenge response for logins

Challenge response protocol for password checking...



When a client wishes to use a resource...

... it requests a connection and negotiates the protocol.

In the reply to this request the server generates and appends an 8 byte, random value - this is stored in the server after the reply is sent and is known as the ***challenge*** - different for every client connection.

The client then uses the hashed password (16 byte values described above), appended with 5 null bytes, as three 56 bit DES keys, to encrypt the challenge 8 byte value, forming a 24 byte value known as the ***response***. This calculation is done on *both* hashes of the user's password, and *both* responses are returned to the server, giving two 24 byte values.

Case study 2: Challenge-response protocol

The server then...

... reproduces the above calculation, using its own value of the 16 byte hashed password and the challenge value that it kept during the initial protocol negotiation.

It then checks to see if the 24 byte value it calculates matches the 24 byte value returned to it from the client. If these values match exactly, then the client knew the correct password and is allowed access.

There are good points about this:

- The server never knows or stores the *cleartext* of the users password - just the 16 byte hashed values derived from it.
- The cleartext password or 16 byte hashed values are never transmitted over the network - thus increasing security.

Case study 2: Challenge-response protocol

However, there is also a bad side:

- The 16 byte hashed values are a "password equivalent". You cannot derive the users password from them, but they can be used in a modified client to gain access to a server.
- The initial protocol negotiation is generally insecure, and can be hijacked in a range of ways. One common hijack involves convincing the server to allow clear-text passwords.
- Despite functionality added to NT to protect unauthorized access to the SAM, the mechanism is trivially insecure

Attacks on Windows (NT) systems...

Even *without* network access, it is possible by various means to access the SAM password hashes, and *with* network access it is easy.

The attack considered here is the use of either a dictionary, or brute force attack directly on the password hashes (which must be first collected somehow).

Case study 3: Kerberos/Cerberus

What is it?

Kerberos is a network ***authentication*** protocol. It provides strong authentication for client/server applications using symmetric or public key cryptography.

Kerberos is freely available in source form, and also is in commercial products (Windows uses Kerberos for login).

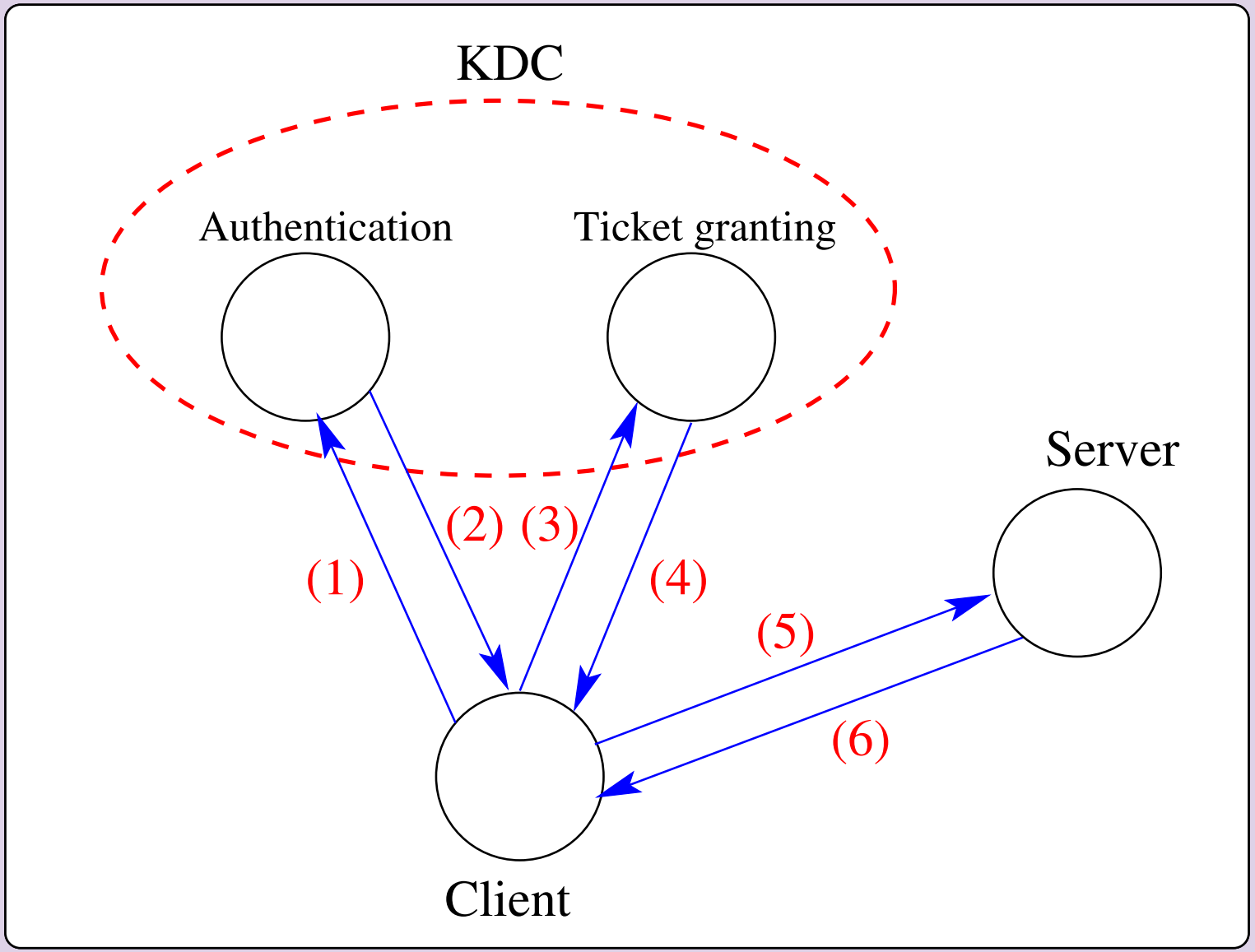
Authentication... and encryption.

Client and server use Kerberos to prove their identity, and encrypt all of their communications. There is a Key Distribution Center (KDC)

Kerberos uses a Needham-Schroeder symmetric key protocol.

Case study 3: Kerberos

Sequence of messages for granting tickets



Case study 3: Kerberos

Notation

A *key* $K_{x,y}$ is a session key shared by both x and y.
When we encrypt a message M using the key $K_{x,y}$ we write it as $E(K_{x,y}, M)$.

Sequence of messages for granting tickets

When a client first authenticates to Kerberos, she:

1. Talks to KDC, to get a *Ticket Granting Ticket* .
2. Uses that to talk to the *Ticket Granting Service* and get a particular ticket for a particular service.
3. Uses that ticket, to interact with the server.

This way a user doesn't have to reenter passwords every time they wish to connect to a Kerberized service.

Note that if the Ticket Granting Ticket is compromised, an attacker can only masquerade as a user until the ticket expires.

Case study 3: Kerberos protocol

Two sorts of credentials: tickets and authenticators:

A *ticket* $T_{c,s}$ contains the client's name and network address, the server's name, a timestamp and a session key,encrypted with the server's secret key (so that the client is unable to modify it).

An *authenticator* $A_{c,s}$ contains the client's name, a timestamp and an optional extra session key, encrypted with the session key shared between the client and the server.

Alice wants session key for communication with Bob:

1. Alice sends message to Ted containing her identity, Ted's TGS identity, and one-time value $(n) : \{a, tgs, n\}$.
2. Ted responds with a key encrypted with Alice's secret key (which Ted knows), and a ticket encrypted with the TGS secret key:
   $E(K_a, \{K_{a,tgs}, n\})\ E(K_{tgs}, T_{a,tgs})$.
   Alice now has ticket and session key: $E(K_{tgs}, T_{a,tgs})$, $K_{a,tgs}$
3. Alice can prove her identity to the TGS, as she has session key $K_{a,tgs}$, and *Ticket Granting Ticket*: $E(K_{tgs}, T_{a,tgs})$.

Case study 3: Kerberos protocol

Later, Alice can ask the TGS for a specific service ticket:

1. When Alice wants a ticket for a specific service (say with Bob), she sends an *authenticator* along with the *Ticket Granting Ticket* to the TGS: $E(K_{a,tgs}, A_{a,b})\ E(K_{tgs}, T_{a,tgs}), b, n$.
2. The TGS responds with a suitable key and a ticket: $E(K_{a,tgs}, \{K_{a,b}, n\})\ E(K_b, T_{a,b})$.
3. Alice can now use an authenticator and ticket directly with Bob: $E(K_{a,b}, A_{a,b})\ E(K_b, T_{a,b})$.

Properties of the protocol - weaknesses -

Host security: Kerberos makes no provisions for host security; it assumes that it is running on *trusted* hosts with an *untrusted* network.

KDC compromises: Kerberos uses a principal's password (encryption key) as the fundamental proof of identity.

Salt: This is an additional input to the one-way hash algorithm.

Outline

Case study 4: Voting protocols

A voting protocol is one in which...

...independent systems vote in a kind of election..

We may want to put conditions on this. For example, afterwards we might want to check that the vote was correct, or that each voter only got a single vote. Systems should be corruption-proof.

Voting systems are commonly found in computer systems. For example, if we had multiple servers being used for redundancy, then the servers might vote to see who is in charge.

Too-simple voting systems may be subject to attack. For example, a protocol like first-to-the-post may result in always choosing only one server, or even worse, be tricked by a fast-responder. (Consider the DHCP attack in the “lecture5” videos).

Case study 4: Voting protocols

Voting using public key encryption...

If Alice votes v_A, then she broadcasts:

$$R_A(R_B(R_C(E_A(E_B(E_C(v_A))))))$$

where

1. R_A is a random encoding function which adds a random string to a message before encrypting it with A's public key:

$$R_A(m) \equiv E(K_{A_{\text{pub}}}, m + rand())$$

2. E_A is public key encryption with A's public key: $E_A(m) \equiv E(K_{A_{\text{pub}}}, m)$.

Each voter then engages in a process of signing and decrypting and rebroadcasting one level of every message/vote.

At the end of the protocol, each voter has a complete signed audit trail and is ensured of the validity of the vote.

Case study 4: First round...

One round...

Who	Receives and removes one level...		and sends on (with sigs)...
Alice:	$R_A^1(R_B^1(R_C^1(E_A^1(E_B^1(E_C^1(v_1))))))$	$\rightarrow$	$R_B^1(R_C^1(E_A^1(E_B^1(E_C^1(v_1)))))$
	$R_A^2(R_B^2(R_C^2(E_A^2(E_B^2(E_C^2(v_2))))))$	$\rightarrow$	$R_B^2(R_C^2(E_A^2(E_B^2(E_C^2(v_2)))))$
	$R_A^3(R_B^3(R_C^3(E_A^3(E_B^3(E_C^3(v_3))))))$	$\rightarrow$	$R_B^3(R_C^3(E_A^3(E_B^3(E_C^3(v_3)))))$
Bob:	$R_B^1(R_C^1(E_A^1(E_B^1(E_C^1(v_1)))))$	$\rightarrow$	$R_C^1(E_A^1(E_B^1(E_C^1(v_1))))$
	$R_B^2(R_C^2(E_A^2(E_B^2(E_C^2(v_2)))))$	$\rightarrow$	$R_C^2(E_A^2(E_B^2(E_C^2(v_2))))$
	$R_B^3(R_C^3(E_A^3(E_B^3(E_C^3(v_3)))))$	$\rightarrow$	$R_C^3(E_A^3(E_B^3(E_C^3(v_3))))$
Charles:	$R_C^1(E_A^1(E_B^1(E_C^1(v_1))))$	$\rightarrow$	$E_A^1(E_B^1(E_C^1(v_1)))$
	$R_C^2(E_A^2(E_B^2(E_C^2(v_2))))$	$\rightarrow$	$E_A^2(E_B^2(E_C^2(v_2)))$
	$R_C^3(E_A^3(E_B^3(E_C^3(v_3))))$	$\rightarrow$	$E_A^3(E_B^3(E_C^3(v_3)))$

In the first round (after 3 transfers) - each voter has agreed that their vote has been counted. If not, they do not continue.

To show agreement, voters sign the votes they forward.

Case study 4: Second round...

Then actually discover the votes...

Who	Receives and removes one level...		and sends on (with sigs)...
Alice:	$E_A^1(E_B^1(E_C^1(v_1)))$	$\rightarrow$	$E_B^1(E_C^1(v_1))$
	$E_A^2(E_B^2(E_C^2(v_2)))$	$\rightarrow$	$E_B^2(E_C^2(v_2))$
	$E_A^3(E_B^3(E_C^3(v_3)))$	$\rightarrow$	$E_B^3(E_C^3(v_3))$
Bob:	$E_B^1(E_C^1(v_1))$	$\rightarrow$	$E_C^1(v_1)$
	$E_B^2(E_C^2(v_2))$	$\rightarrow$	$E_C^2(v_2)$
	$E_B^3(E_C^3(v_3))$	$\rightarrow$	$E_C^3(v_3)$
Charles:	$E_C^1(v_1)$	$\rightarrow$	v_1
	$E_C^2(v_2)$	$\rightarrow$	v_2
	$E_C^3(v_3)$	$\rightarrow$	v_3

Only Alice can remove her level of encryption, but anyone can check that it was done correctly (by re-encrypting).

In the second round, someone can tamper with the vote, but at the end each vote can be re-encrypted, and checked against the set of signatures. The tamperer will be found.

Case study 5: Tossing a coin

Lets pretend we are gambling...

Alice and Bob want to toss a coin, so adopt the following first steps:

1. Alice calculates two primes p, q and calculates $N = pq$, sends N to Bob. $N = 35 = 5 * 7$
2. If Bob can factorize the number, then Bob wins a coin toss.
3. Bob selects random x, and sends $y = x^2 \bmod N$ to Alice. $y = 31^2 \bmod 35 = 16$
4. Alice calculates the four square roots of 16:

$$\begin{array}{llll} 4^2 \bmod 35 = 16 & & 31^2 \bmod 35 = 16 \\ 24^2 \bmod 35 = 16 & & 11^2 \bmod 35 = 16 \end{array}$$

This is easy for Alice, as she knows the prime factors of N. She then sends one of these back to Bob.

Case study 5: Tossing a coin

So what does Bob do?

If Bob receives x or $-x$, then he learns nothing, but if Bob receives either of the other values:

$$\begin{aligned} \mathrm{GCD}(24+31,35) &= \mathrm{GCD}(55,35) \\ &= 5 \end{aligned}$$

Bob can calculate a factor, and Alice is unable to tell she has divulged the factor.

We already saw this in Shor's algorithm...

... If you know two different square roots of a number modulo $N = pq$, you can find the factors of N.

Alice can compute square roots knowing p, q?

Yes. When Alice receives the value $y = x^2 \bmod n$, she calculates $\sqrt{y} \bmod p$ and $\sqrt{y} \bmod q$. She then uses these two values in the Chinese Remainder Theorem to find a root. Using the example in class, $y = 16$, and Alice knows $p = 5$ and $q = 7$. She computes $\sqrt{16} \bmod 5$ and $\sqrt{16} \bmod 7$, noting that $16 = 1 \bmod 5$ and $16 = 2 \bmod 7$:

$$\begin{aligned} \sqrt{1} \bmod 5 &= \pm 1 \\ \sqrt{2} \bmod 7 &= \pm 4 \end{aligned}$$

and then CRT helps find x:

$$\begin{aligned} x \bmod 5 = 1 \;\&\; x \bmod 7 = 4 &\implies x = 11 \\ x \bmod 5 = 1 \;\&\; x \bmod 7 = -4 &\implies x = 31 \end{aligned}$$

We have the four square roots of 16 mod 35: ± 11 and ± 31.

Example of a square root algorithm modulo an odd prime...

If $p \equiv 3 \bmod 4$ then $x = y^{\frac{p+1}{4}} \bmod p$.

For example, if $p = 3851$ and $y = 3298$, then $x = 3298^{\frac{3852}{4}} \bmod 3851 = 231$.

Case study 6: Contracts, and oblivious transfer

Scenario: signing contracts can be difficult:

If one party signs the contract, the other may not. We have one party bound by the contract, and the other not.

In addition, both may sign, and then one may say "*I didn't sign any contract!*" afterwards.

Use oblivious transfer...

In an oblivious transfer, randomness is used to convince participants of the fairness of some transaction

---

In the coin-tossing example, Alice knows the prime factors of a large number, and if Bob can factorize the number, then Bob wins a coin toss. Alice will either divulge one of the prime factors to Bob, or not, with equal probability. Alice is unable to tell if she has divulged the factor, and so the coin toss is fair.

Alice is "oblivious" - she is unaware if she has revealed a secret.

Case study 6: Contract signing

Use oblivious transfer for contract signing...

Oblivious transfer used for contract-signing where the following points are important:

- Up to a certain point neither party is bound
- After that point both parties are bound
- Either party can prove that the other party signed

Alice and Bob pre-agree on some probability difference that they will live with (say 5%), and then exchange (say 1000) signed oblivious-transfer messages, each possibly transfering a bit of information. They are becoming "more" bound by a contract with ever-increasing probability - perhaps each successfully received bit is worth 1% to the receiver.

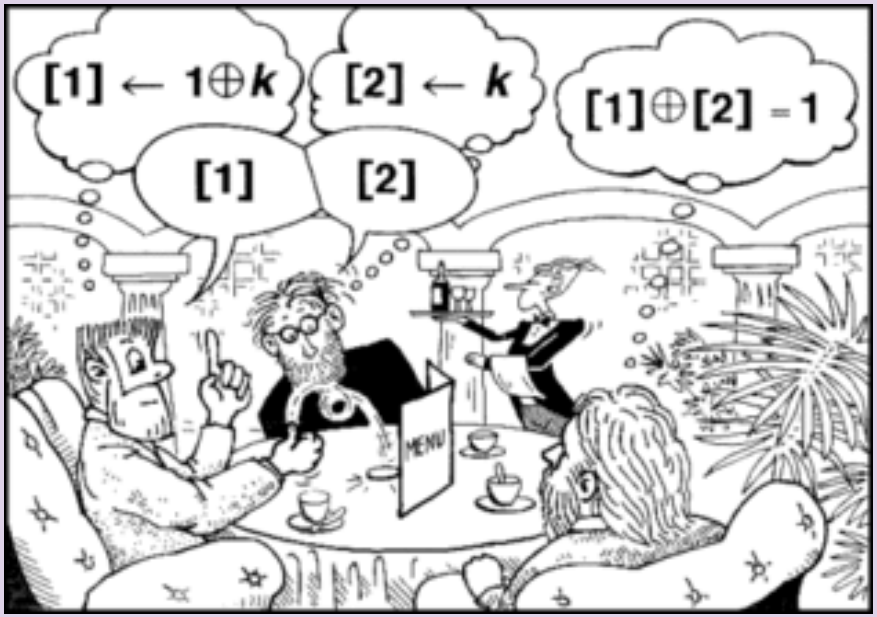
What if someone tries to terminate early?

In the event of early termination of the contract, either party can take the messages they have to an adjudicator, who chooses a random probability value (42% say) before looking at the messages.

If messages are over 42% then both parties are bound. If less then both parties are free.

Case study 7: Anonymity protocol

The Dining Cryptographers...



Consider a group of (3 or more) cryptographers, dining at a fancy restaurant. When they ask for the bill, the waiter tells them that the meal has been paid for, by someone who wishes to remain anonymous. The only other person who had been in the restaurant was from the NSA, so the cryptographers want to find out if the NSA guy paid, or if one of their own group paid.

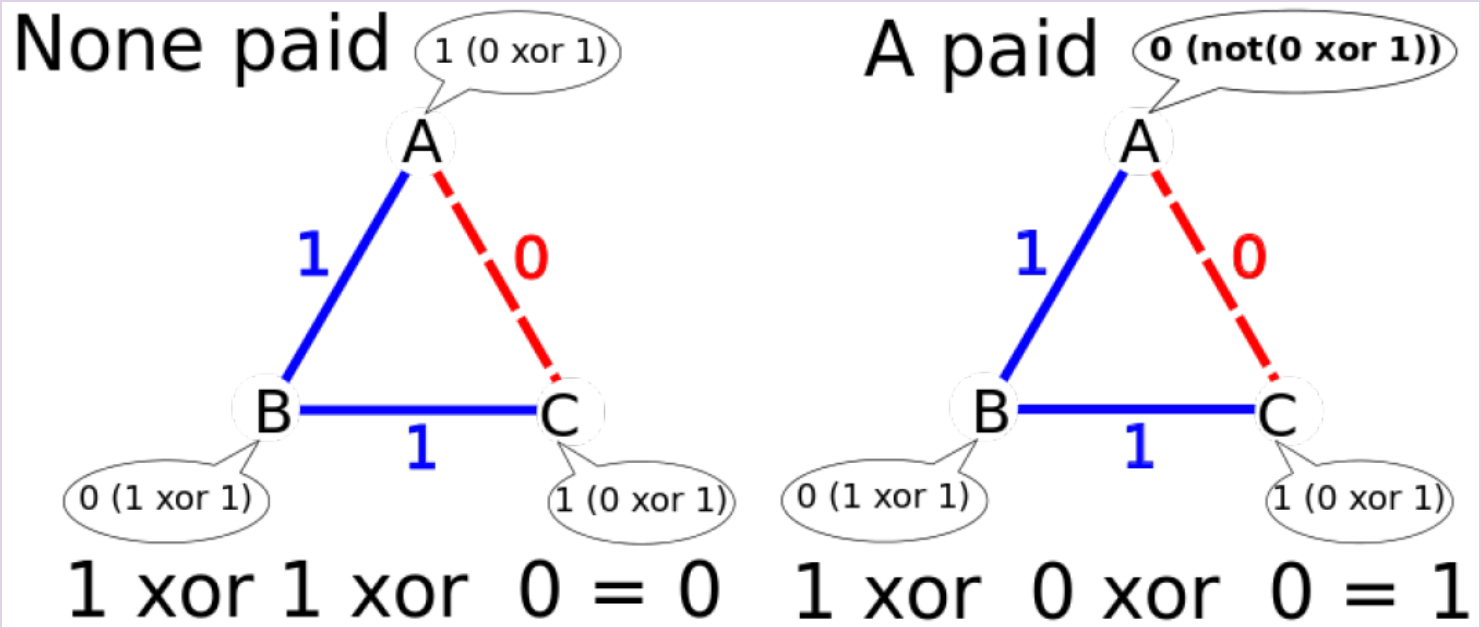
Can they devise a protocol which lets them find out if one of their own group paid for the meal?

Case study 7: Anonymity protocol

The Dining Cryptographers...

Firstly, each pair of adjacent cryptographers decide on a randomly chosen bit (0/1) behind the menu so no-one else can see. They each remember the bit (0/1) they share with each neighbour.

If they did not pay for the meal, they announce the XOR of their two neighbours. Otherwise they announce the reverse.



The parity of the values is 1 if one of them paid for the meal.

Case study 7: Anonymity?

Why have one?

In the example, we have a bit of information, but we end up with no way of finding out from whom it came, unless somehow all the coin flips are recorded.

We can scale up the method, to anonymize any number of bits. Applications for this technique might include

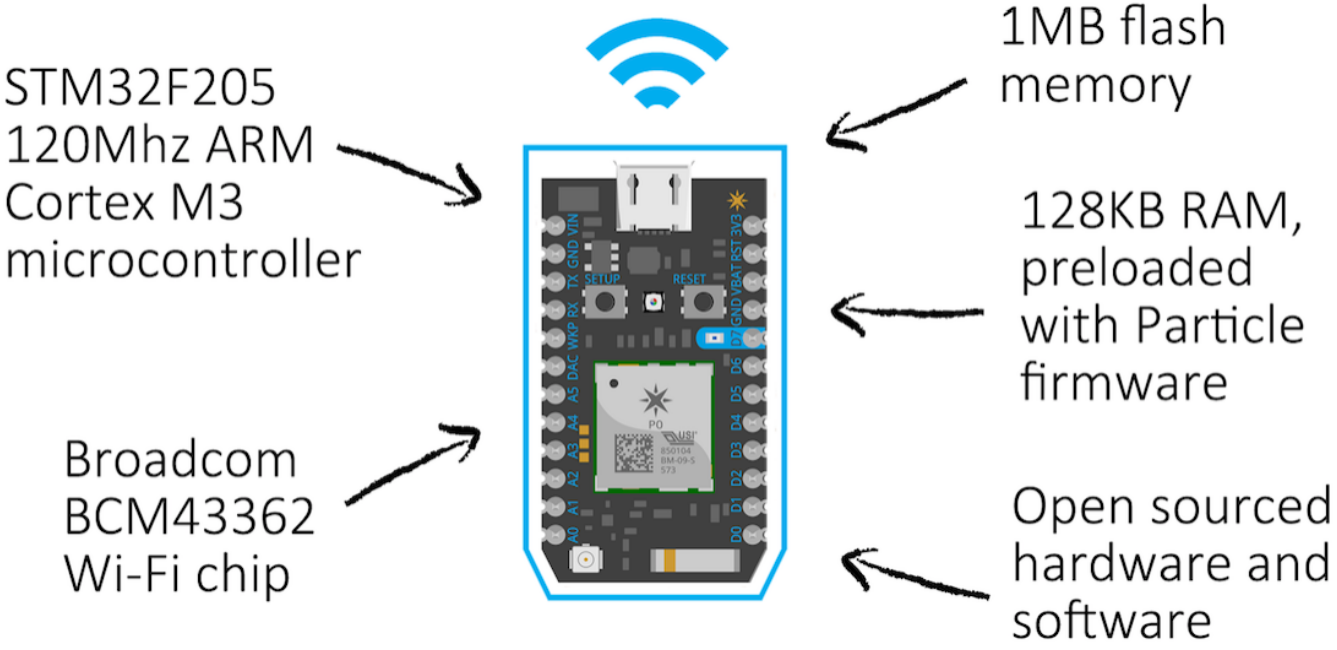
1. Anonymous email - perhaps for a whistle-blower to use.
2. Anonymizing data sources of sensitive information, to ensure the data is supplied without fear of retribution.

There are anonymizing networks, which provide supposedly anonymous Internet communication. For example the Tor network is (amongst other things) an Onion router. In an Onion router, messages are repeatedly encrypted and then each onion router removes a layer of encryption to uncover routing instructions, and sends the message to the next onion router and so on. This prevents intermediary nodes from knowing the origin, destination, and contents of the message.

Outline

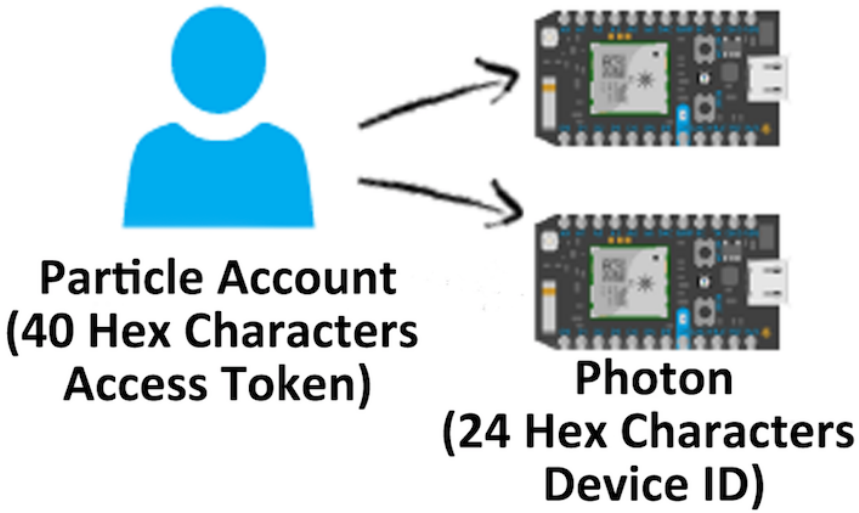
A well known IoT device...

The **Particle Photon** is a popular IoT development kit that is relatively inexpensive, at US$19.



Single token needed for access...

Users can associate multiple Particle devices to their Particle account. The access token is required to use the Cloud API provided by Particle and it is considered secret and should not be shared.



Topic 1: Accessing the IoT device

The device access is via the "Particle cloud API"

Whenever a user sends an instruction to the photon, the user has to supply a device ID, and the access token. The Particle cloud will authenticate the token and device ID before relaying instructions to the device.

The access token is all-powerful! With the token, you can list out all the device IDs. i.e. you do not need to know the device ID to send instructions. As a result, getting the access token ... gets you the photon!

Acquiring the access token...

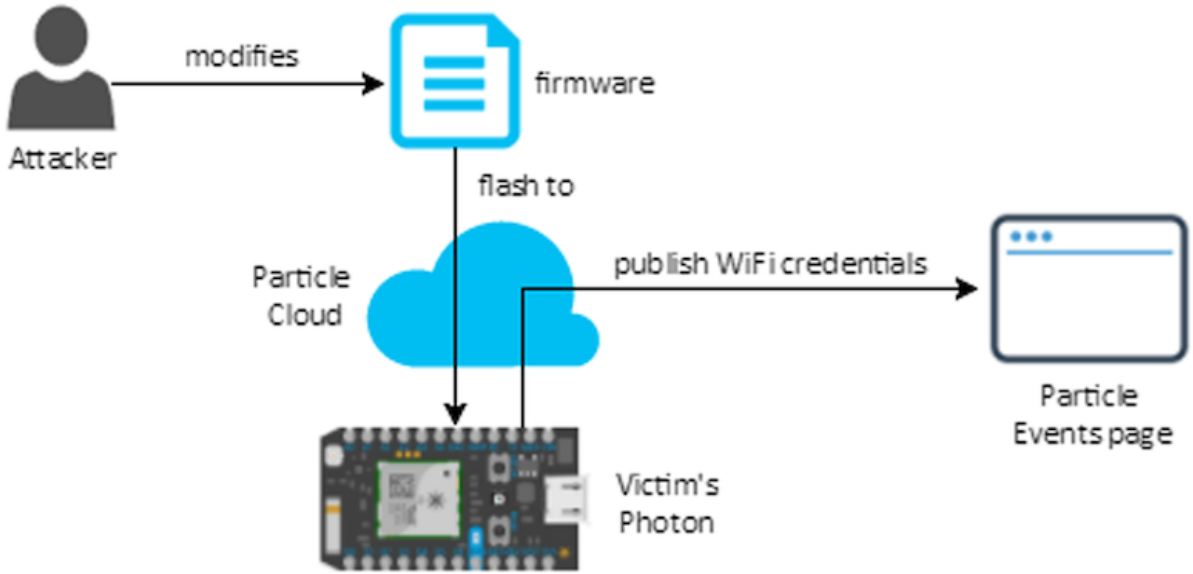
Students identified various mechanisms to get an access token:

- Social Engineering
- Man-in-the-middle attacks

They were effective mechanisms, and could be used to get the access token for any number of photons, nearby or remote. The students even found access tokens through google!

Single token needed for access...

An attacker can **build a modified firmware** that **publishes the stored Wi-Fi credentials**. The attacker can then flash the firmware using the Particle Cloud onto the Photon device. Firmware flashing takes only a short while and **does not overwrite the current user application** installed on the device. This allows the attacker to compromise the Photon device **without disrupting its usual functions.**



Topic 1: An attack on IoT and photons

My students engineered this nasty attack

And they are talking to the manufacturers to come up with a resolution. What can be done to mitigate it?

- Limit the access token capabilities.
- Two factor authentication
- Authenticate the clients who initiate software updates...

The firmware steals WiFi credentials for the remote network near the photon. This is obviously pretty serious.

.

BAD STUDENTS!

Topic 1: IoT

Poster and papers, responsible disclosure...

IoT Hacking: Attacking the [redacted]

Ang Kiang Siang, National University of Singapore, 21 Lower Kent Ridge Rd, 119077, +65 90495036, a0125528@u.nus.edu

Chew Yung Chung, National University of Singapore, 21 Lower Kent Ridge Rd, 119077, +65 90495036, a0133662@u.nus.edu

Tan Qiu Hao, Joel, National University of Singapore, 21 Lower Kent Ridge Rd, 119077, +65 90495036, a0125473@u.nus.edu

Juliana Seng, National University o Singapore, 21 Lower Kent Ridge f 119077, +65 90495036, a0126332@u.nus.ed

ABSTRACT

The Internet of Things (IoT) is becoming increasingly prevalent as the world moves towards the vision of a smarter home and city. As more of these products, which range from interconnected sensors to wearable devices, get connected to the Internet, serious concerns are being raised with regards to its security. Unfortunately, as the domain of IoT is relatively new, security is inadequately considered in these devices. This paper explores the security aspects of a popular IoT device, the [redacted]. Furthermore, the paper demonstrates a viable attack to compromise a [redacted] and retrieve its stored Wi-Fi passwords over the Internet. Finally, the paper proposes potential defense measures that can be adopted to better secure the [redacted].

Categories and Subject Descriptors

C.2.1 [**Computer-Comm. Networks**]: Network Architecture and Design; C.2.3 [**Computer-Comm. Networks**]: Network Operations; C.2.6 [**Computer-Comm. Networks**]: Internetworking; C.3 [**Special-purpose and application based systems**]

General Terms

Security.

Keywords

IoT, [redacted]

1. INTRODUCTION

Internet of Things (IoT) refers to the interconnected network of physical objects that enables the collection and exchange of data over the Internet. Examples include using IoT to monitor and manage energy consumption within a household, remotely switching on/off appliances, and automatically unlocking of doors. Contributing to the revolution of smart cities worldwide, advanced Information and Communication Technologies (ICTs) infrastructures have implemented IoT as one of the key technologies supporting this growth.[6] This shows the importance of IoT in today's market and how it will be widely used in future. Furthermore, with this increased involvement of IoT on critical data over the Internet, questions on the security of IoT devices should be raised and investigated. In 2014, a researcher in Proofpoint uncovered a botnet where 25% of the traffic comprised of compromised IoT devices [5]. This incident demonstrated the potential risk of unsecured IoT devices, and further affirm importance of IoT security.

Out of all of the IoT devices in the market, the [redacted] was chosen because the device is a popular IoT developme and is relatively inexpensive, at US$19. The [redacted] henceforth [redacted] is a Wi-Fi enabled development kit mea IoT prototyping. The [redacted] is a product by [redacted] and is of a family of development kits, including kits with ce connectivity. The [redacted] leverages on another product by a [redacted] chip, which contains a STM32F205 120Mhz ARM C M3 microcontroller and a Broadcom BCM43362 Wi-Fi chi communicates on 802.11b/g/n protocol. The [redacted] also c with a 1MB flash memory and 128KB RAM which are prel with [redacted] firmware libraries. Because [redacted] product meant for prototyping and subsequent deployment, the [redacted] both open sourced hardware and software.

In this paper, the security aspects of the [redacted] will be expl Additionally, a viable attack to compromise a [redacted] retrieve its stored Wi-Fi passwords over the Internet wi demonstrated. Finally, the paper will propose potential de measures that can be adopted to better secure the [redacted]

2. BACKGROUND

2.1 [redacted] Platform

Before the [redacted] features can be accessed, a user has to si for a [redacted] account. Upon signup, the account is entitled a to the [redacted] Platform, which include tools such as the web mobile SDK, and the device dashboard.

2.2 Access Token and Device ID

Each [redacted] account is assigned an access token, a 40 hexadecimal string that is used to interact with the account a associated devices through the use of [redacted]'s Cloud API default, the assigned access tokens do not expire, but user allowed to assign an expiry date to them.

IoT Hacking: Attacking the Photon

Ang Kiang Siang, National University of Singapore, 21 Lower Kent Ridge Rd, 119077, +65 90495036, a0125528@u.nus.edu

Chew Yung Chung, National University of Singapore, 21 Lower Kent Ridge Rd, 119077, +65 90495036, a0133662@u.nus.edu

Tan Qiu Hao, Joel, National University of Singapore, 21 Lower Kent Ridge Rd, 119077, +65 90495036, a0125473@u.nus.edu

Juliana Seng, National University of Singapore, 21 Lower Kent Ridge Rd, 119077, +65 90495036, a0126332@u.nus.edu

ABSTRACT

The Internet of Things (IoT) is becoming increasingly prevalent as the world moves towards the vision of a smarter home and city. As more of these products, which range from interconnected sensors to wearable devices, get connected to the Internet, serious concerns are being raised with regards to its security. Unfortunately, as the domain of IoT is relatively new, security is inadequately considered in these devices. This paper explores the security aspects of a popular IoT device, the Particle Photon. Furthermore, the paper demonstrates a viable attack to compromise a Particle Photon, and retrieve its stored Wi-Fi passwords over the Internet. Finally, the paper proposes potential defense measures that can be adopted to better secure the Particle Photon.

Categories and Subject Descriptors

C.2.1 [**Computer-Comm. Networks**]: Network Architecture and Design; C.2.3 [**Computer-Comm. Networks**]: Network Operations; C.2.6 [**Computer-Comm. Networks**]: Internetworking; C.3 [**Special-purpose and application based systems**]

General Terms

Security.

Keywords

IoT, Particle, Photon.

1. INTRODUCTION

Internet of Things (IoT) refers to the interconnected network of physical objects that enables the collection and exchange of data over the Internet. Examples include using IoT to monitor and manage energy consumption within a household, remotely switching on/off appliances, and automatically unlocking of doors. Contributing to the revolution of smart cities worldwide, advanced Information and Communication Technologies (ICTs) infrastructures have implemented IoT as one of the key technologies supporting this growth.[6] This shows the importance of IoT in today's market and how it will be widely used in future. Furthermore, with this increased involvement of IoT on critical data over the Internet, questions on the security of IoT devices should be raised and investigated. In 2014, a researcher in Proofpoint uncovered a botnet where 25% of the traffic comprised of compromised IoT devices [5]. This incident demonstrated the potential risk of unsecured IoT devices, and further affirms the importance of IoT security.

Out of all of the IoT devices in the market, the Particle Photon was chosen because the device is a popular IoT development kit and is relatively inexpensive, at US$19. The Particle Photon, henceforth Photon, is a Wi-Fi enabled development kit meant for IoT prototyping. The Photon is a product by Particle, and is a part of a family of development kits, including kits with cellular connectivity. The Photon leverages on another product by Particle, a P0/P1 chip, which contains a STM32F205 120Mhz ARM Cortex M3 microcontroller and a Broadcom BCM43362 Wi-Fi chip that communicates on 802.11b/g/n protocol. The Photon also comes with a 1MB flash memory and 128KB RAM which are preloaded with Particle's firmware libraries. Because Particle products are meant for prototyping and subsequent deployment, the Photon has both open sourced hardware and software.

In this paper, the security aspects of the Photon will be explored. Additionally, a viable attack to compromise a Particle Photon and retrieve its stored Wi-Fi passwords over the Internet will be demonstrated. Finally, the paper will propose potential defense measures that can be adopted to better secure the Particle Photon.

2. BACKGROUND

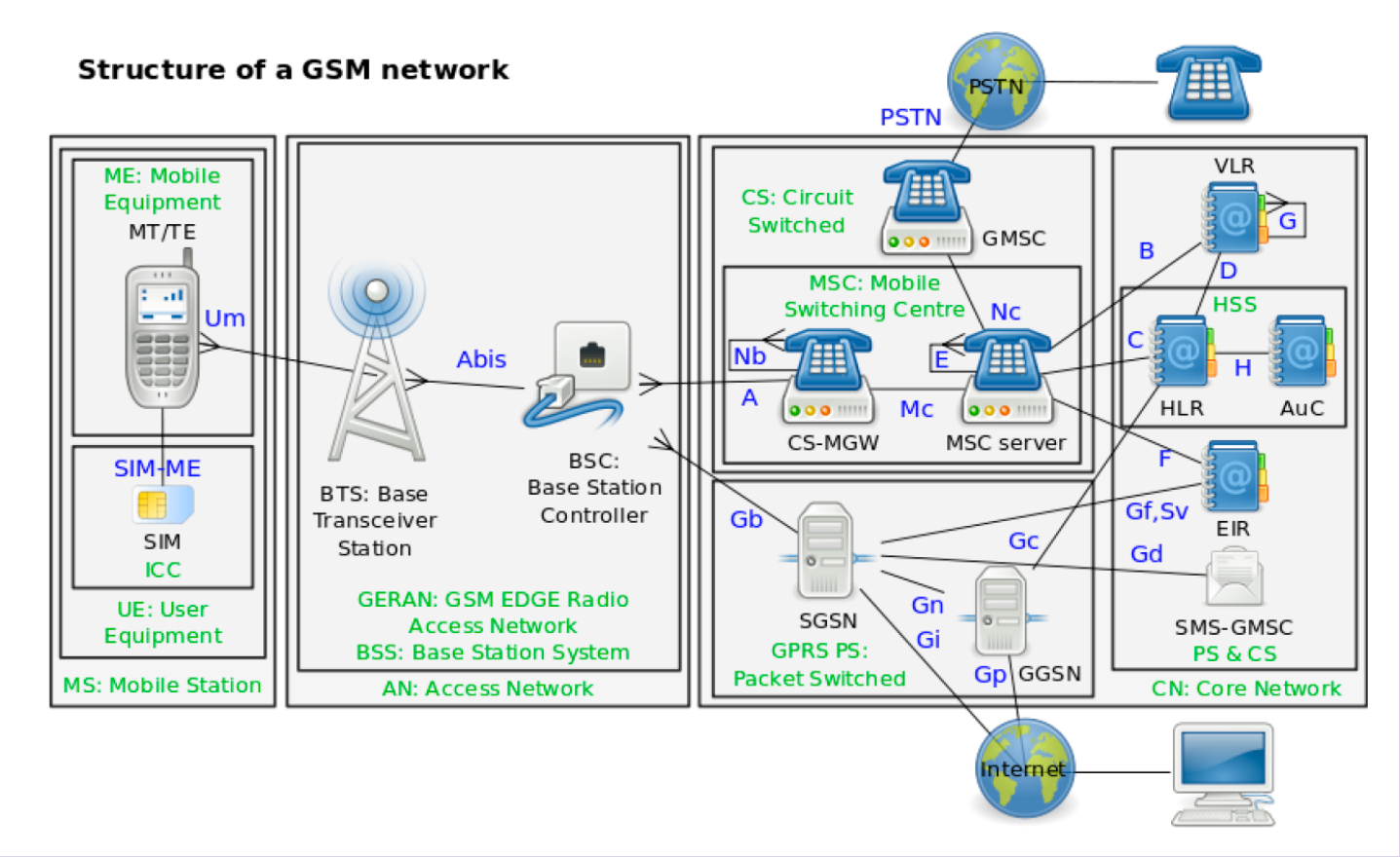
2.1 Particle Platform

Before the Photon's features can be accessed, a user has to sign up for a Particle account. Upon signup, the account is entitled access to the Particle Platform, which include tools such as the web IDE, mobile SDK, and the device dashboard.

2.2 Access Token and Device ID

Each particle account is assigned an access token, a 40 digit hexadecimal string that is used to interact with the account and its associated devices through the use of Particle's Cloud API.[3] By default, the assigned access tokens do not expire, but users are allowed to assign an expiry date to them.

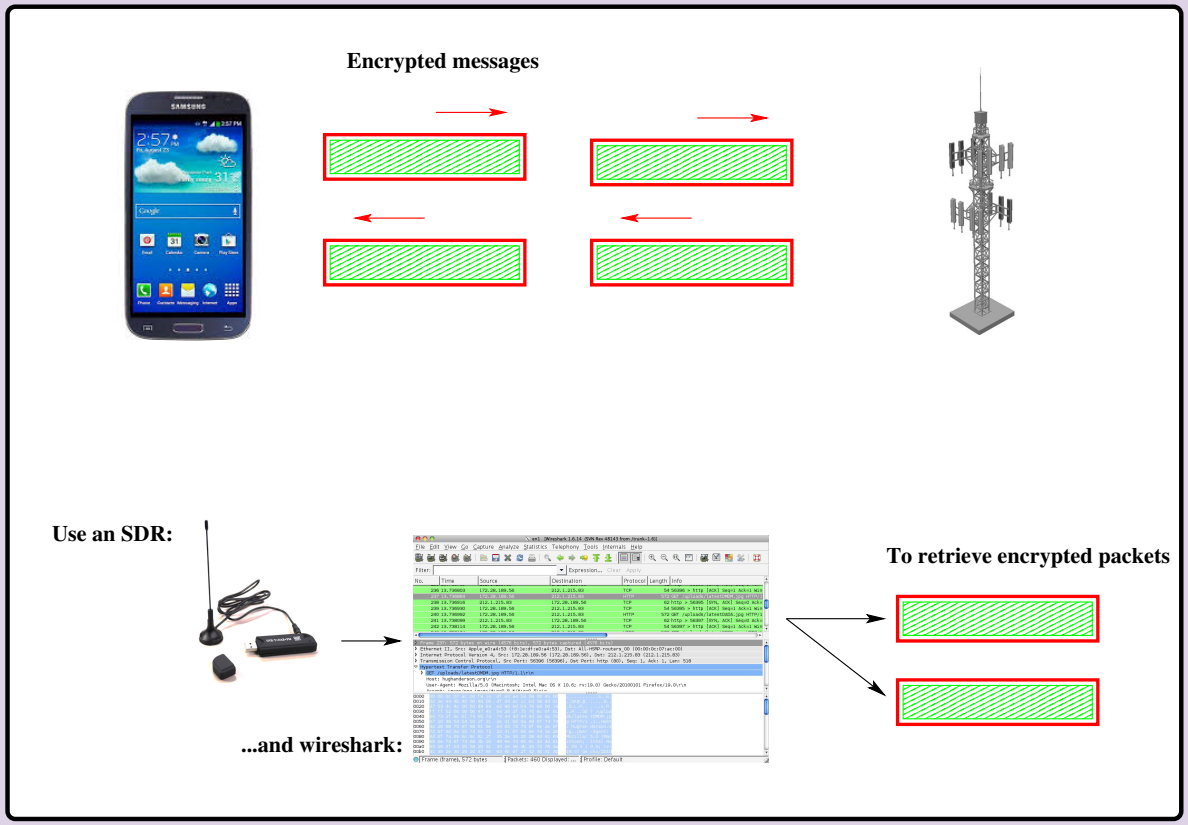
Mobile stations with SIM...



The SIM keeps the private identity of a phone...
Base stations authenticate the SIM/phone.
GSM phone networks on 900/1800 MHz - can we look at them with an SDR?

Topic 2: An attack on phones and GSM

Step 1: get the packets/frames off the air.....



The packets/frames are 124bits, and some are always the same.
In particular, my students found that in Singapore, a “system information type 6” message was found always 306 frames away from another fixed frame.
The decoded frame was all 0x2b!

Topic 2: An attack on phones and GSM

If the plaintext is known...

... but the key is not, this is like a hash function. We have seen it before. What can we do?

- Keysize is large (128 bits), but build a rainbow table. The one we have was built in Germany a few years ago, and is 2TB.
- The rainbow table, with good probability, discovers the input to the hash function i.e. the key!

As a result, we have the key used by the phone, and are able to decode this session - the students successfully decrypted SMS messages.
They were not able to do voice, because the simple SDR they were using could only receive a single channel at a time.

Topic 2: Another attack - GSM man-in-the-middle

Using software defined radios...

The SDR on the right is pretending to be a base station, the phones on the left can see it... With some care, you can make the phones associate to the SDR instead of Singtel/Starhub, and retrieve interesting information. Students developed a social engineering SMS sequence...

Outline

Secrets come in lots of flavours

Not just secret keys...

We have seen how secrets can be useful, particularly for keeping keys for cryptography.

Secrets may also be algorithmic...

- An encryption algorithm
- A particular protocol (sequence of communications).
- A hashing algorithm.

These secrets are programs, not data. How do we keep these sorts of secrets?

Applications running on PCs, phones, ...

Consider the compilation process...

Often, the executable is distributed, not the source. RMS points out many reasons why this is most likely a bad idea.

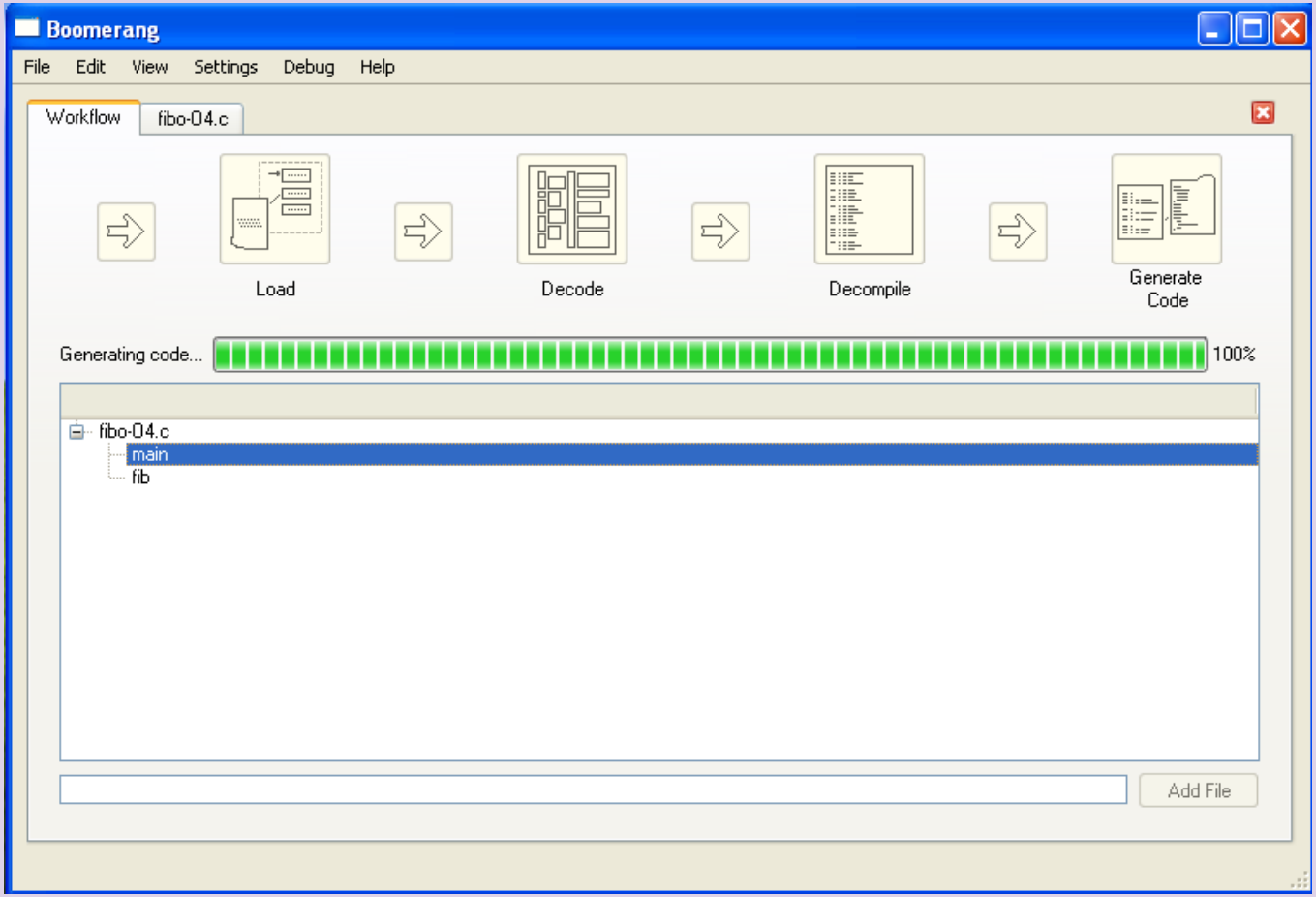
But, in a *security* scenario, you may think that by only distributing your executable code, you can hide your source code *secrets*.

This is a silly idea. You should keep no secrets in source code.

- You should not have source code secrets (remember the notion of open design), and
- you cannot hide things by using compilation.

Why?

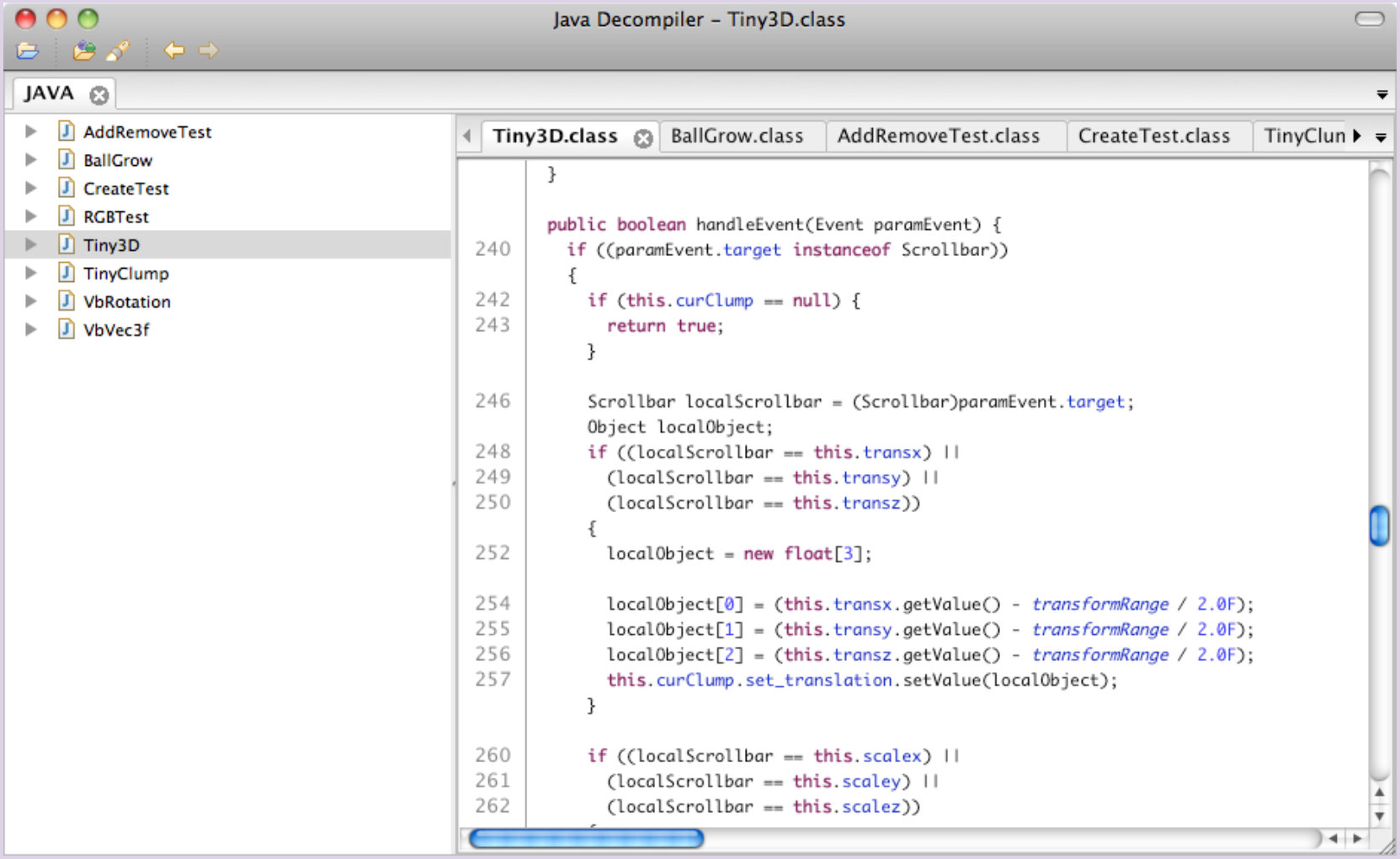
Decompilation is reverse of compilation...



Note that really good decompilation may take user input, but it is possible.

Java...

Java decompilation is particularly easy...



Applications...

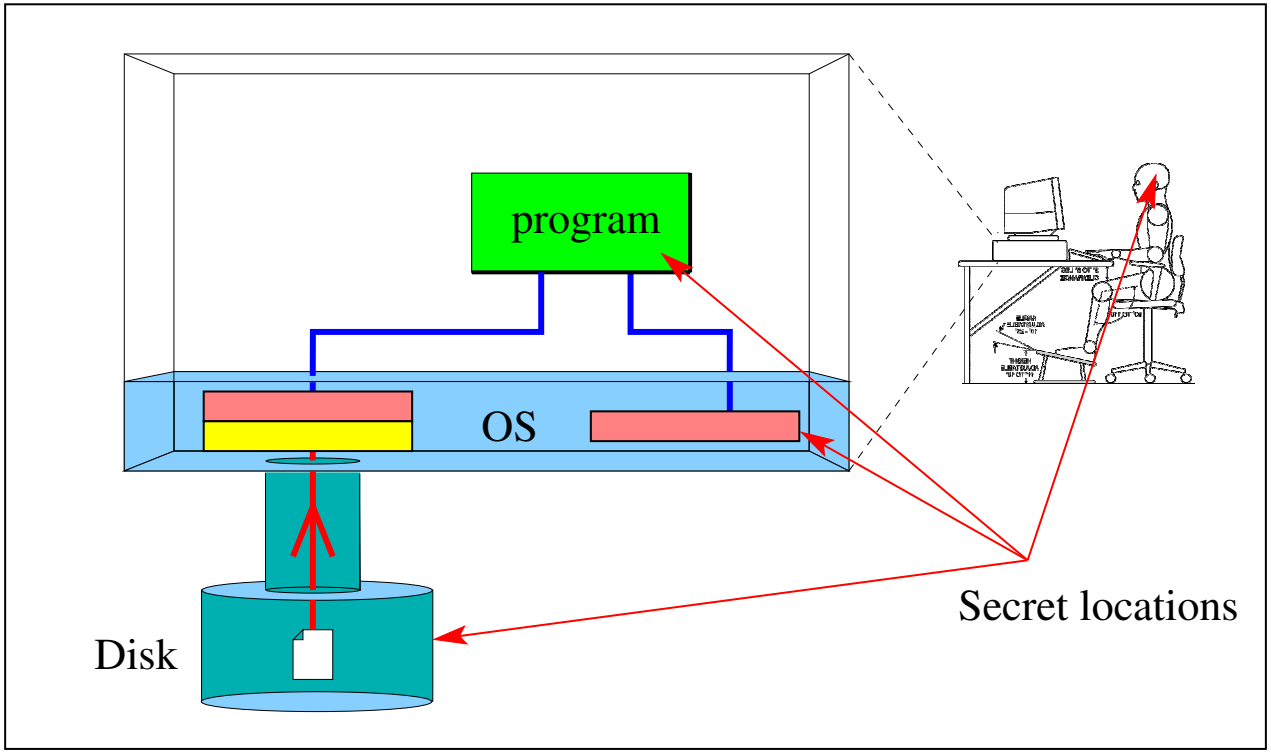
...are, essentially, an open book...

If the platform is sufficiently complex (e.g, Windows, GNU+Linux, Android, Symbian), then it is likely that sources for any application could be generated easily, so if you intend to distribute your application to other users, and it relied on some secret in the code, that secret is unsafe.

Of course - you can still have secrets. They must be kept separately. (Do I detect an example of **Least common mechanism:** Minimize the amount of mechanism common to more than one user and depended on by all users?)

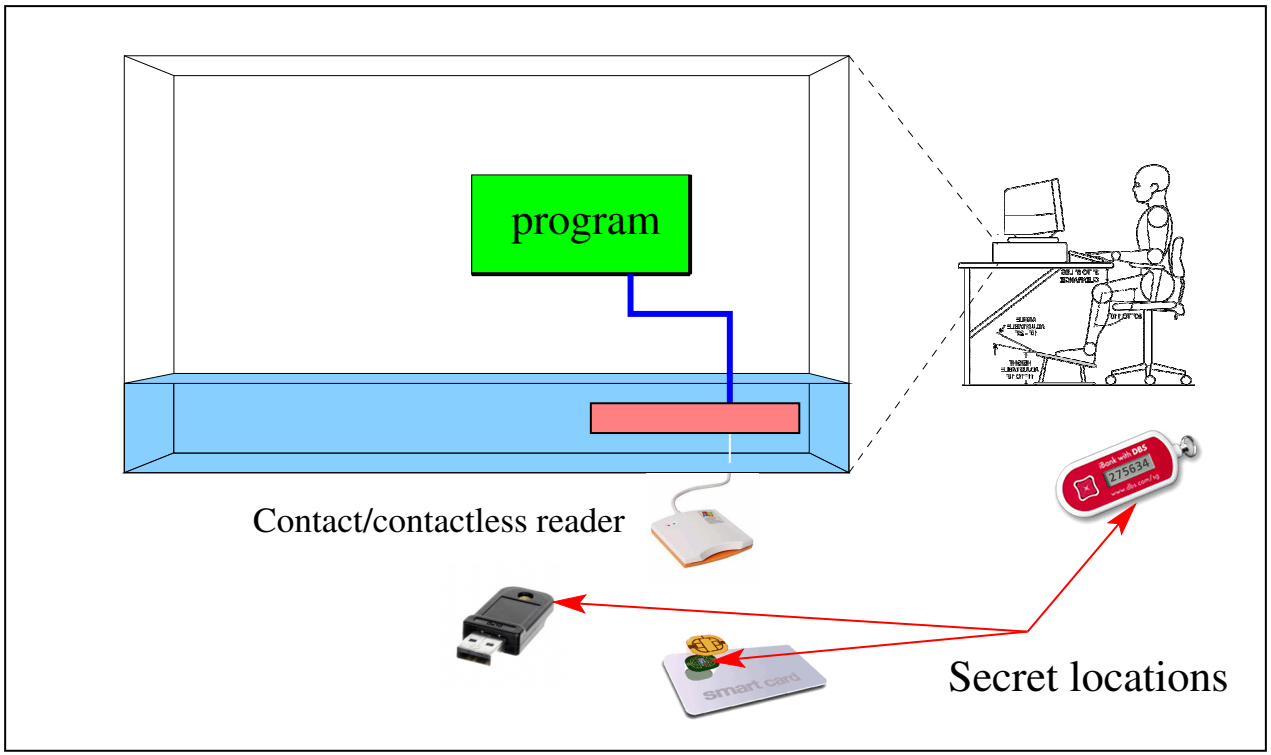
Possible locations for secrets

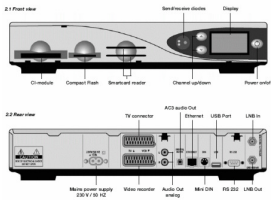
Disk (file), program, OS, brain...

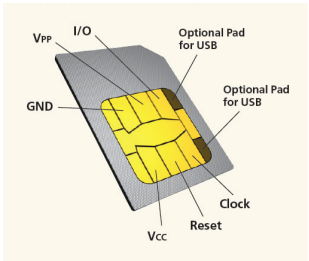


Other locations for secrets

Dongles, smart cards, 2-factor devices...







Lets get the PC out of the way first...

There is no such thing as a safe PC...

Secrets may be data, or algorithmic.

Complex platforms are unsuitable for hiding secrets - they are open to attacks:

- physical (someone breaks open a box and steals a disk containing secrets), or
- software based (someone manages to install or otherwise hack either an application or a whole OS).

A strategy for getting around this is to somehow install the secret on a *portable* computer system that is harder to attack (more resistant to tampering), and possibly, resistant to theft.

- Examples of this approach are found everywhere (smart cards, SIM cards...).
- Such external devices typically contain small computers with small amounts of memory.

Classes of attackers for security hardware

For tamper-resistant portable devices...

Anderson outlines this classification:

- Class 1 attackers: Clever outsiders, who may not have detailed knowledge of the inner workings of the system
- Class 2 attackers: Insiders, with detailed system knowledge.
- Class 3 attackers: Governments, the Mafia - who may have money and time to burn. May also have strong motivation.

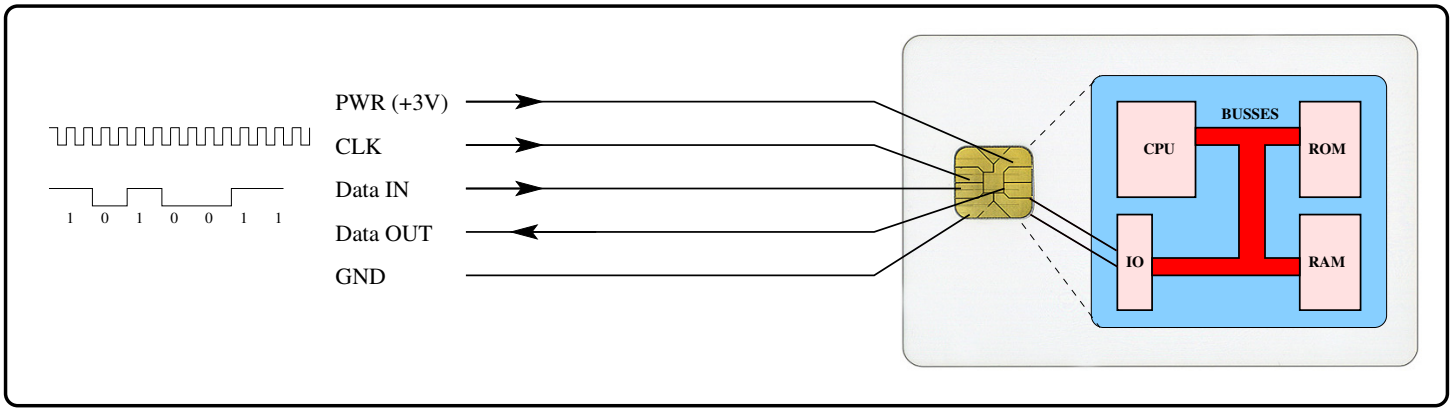
Anderson points out that persistence and cunning are some of the hallmarks of an effective attacker, and that sometimes the class 1 attacker can do things that the class 3 attacker failed to do.

We have both invasive and non-invasive attacks:

- Invasive: the semiconductor chip is decapsulated, and direct attacks made on the circuitry. Some of these attacks require expensive equipment.
- Non-invasive: we manipulate the device without decapsulation. May need only inexpensive equipment.

Typical hardware on a Smart/SIM card

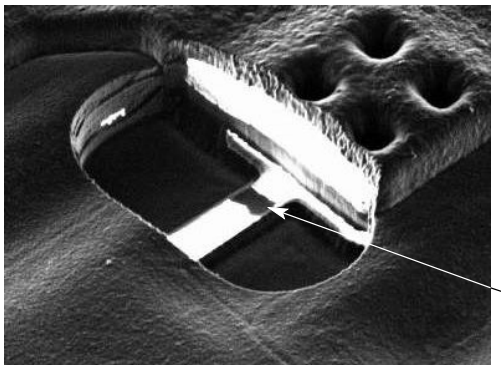
A high-security Smart/SIM card controller



- CPU, memory and IO are under the (gold) connector. They are all in one chip, and the components are connected by (internal) busses.
- Reader must supply power and a clock to operate the CPU.
- Only external signaling is a single line IN and a single line OUT.
- Data bits are serialized. 1010011 above might correspond to the byte (hex) 0x53, perhaps corresponding to the ASCII letter S.

Fuses: sample security-specific feature

For devices like these...



The chips come from the factory all the same, and need to be programmed.

- When the devices are manufactured, the memory (PROM) can be read and written from outside using the serial port.
- Once the device has been uniquely programmed, a fuse is blown. This fuse is just a track in the IC, and is blown by providing too much current.

A blown fuse means no more (external) reading and writing.

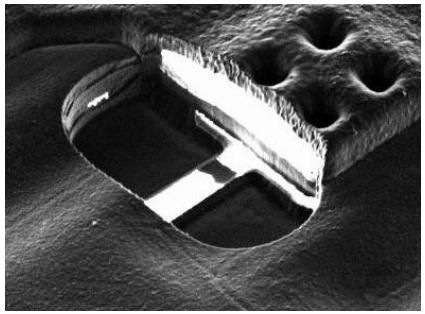
Fuses: Class 3 attacker hack: rewiring

Assuming you happen to have some fancy equipment

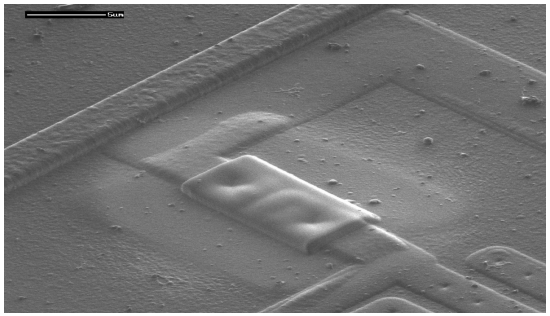
Recipe: Take a common household electron microscope, a focused ion beam tool, and various other items. And then...

- Expose chip using chemical and/or laser cutting (decapsulation).
- Re-connect the fuse using deposition or tiny probes.
- Read the memory using the serial port.

Decapsulation
(Nitric acid...)



Discover blown fuse
(May need FIB to
expose layer)



Deposit new material over fuse

Fuses: Class 1 attacker (the outsider) hack 1

Only a little equipment needed...

- The outsider does not have electron microscopes or FIB units, or even Nitric acid! Life is tough sometimes.
- The outsider notices that when power is first applied, the smart card outputs an identification string on the serial OUT line:

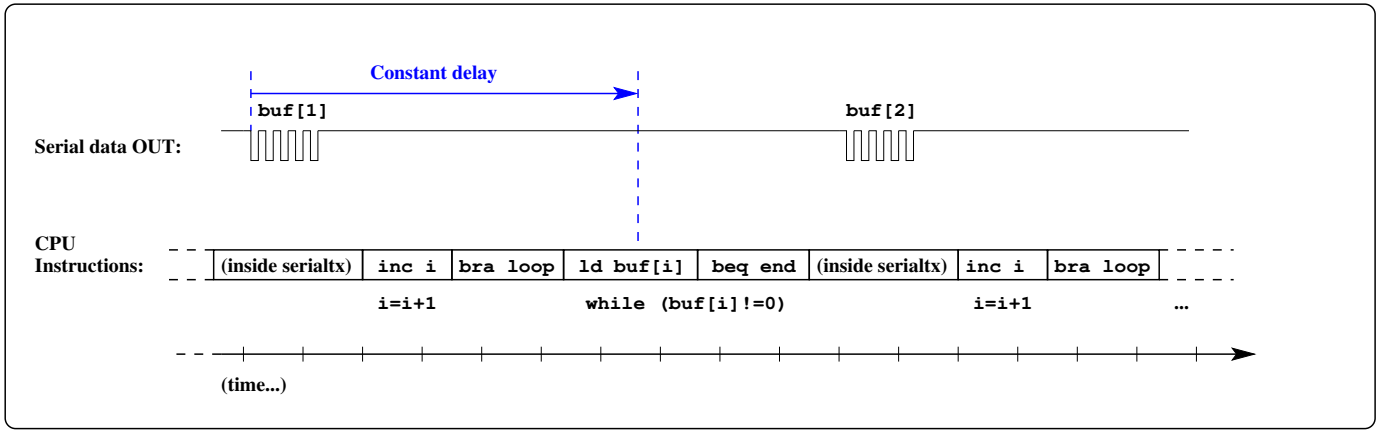
```
Vsn 0.97
```

- The outsider knows that the software is very small, and probably stores strings like “Vsn 0.97” in 9 bytes of memory, the 8 bytes of the string, terminated by a null (a byte with all zeroes in it’s bits).
- The outsider surmises that the code is probably like this:

```
void printit( char buf[] ) {
   int i=0;
   while ( buf[i]!=0 ) {
      serialtx(buf[i] );
      i=i+1;
   }
}
...
   printit( “Vsn 0.97” );
```

Fuses: Class 1 attacker hack 1

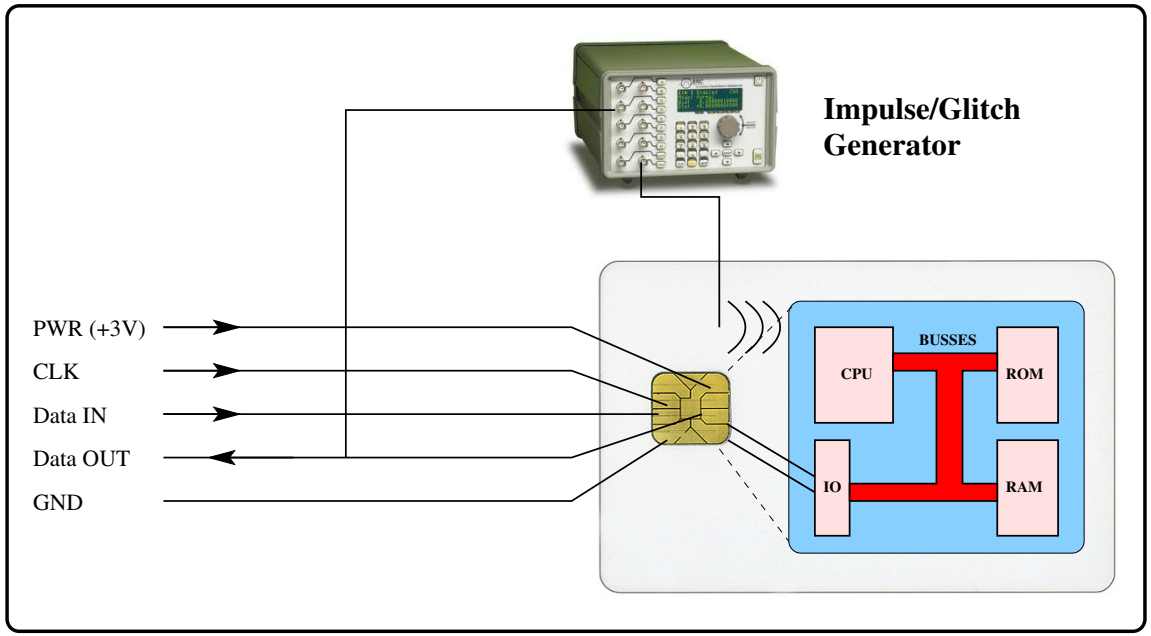
Timing diagram for the printit() while loop...



- The instructions executed by the CPU are predictable. After transmitting the serial character for `buf[1]`:
  - The `i` variable is incremented (`inc i`)
  - The CPU branches to the beginning of the while loop (`bra loop`)
  - The value in `buf[i]` is loaded (`ld buf[i]`)
  - If it is equal to 0, then branch to the end of the loop (`beq end`)
  - Otherwise transmit the next character...
- There is a constant delay between the character and these instructions.

Fuses: Class 1 attacker hack 1: glitch

A "glitch" attack...



- By timing a pulse/glitch to occur exactly when the software checks the test at the beginning of the while loop, (it is always a fixed time after the previous character), attacker can make the test always succeed.
- i.e. `buf[i]!=0` will always be `TRUE`.
- A glitch may be a spark, a variation of the power supply, or some other activity. Hardware to do this is cheaply available (e.g. in most EE labs).

Fuses: Class 1 attacker hack 1: glitch

A "glitch" attack...

- If `buf[i]!=0` is always `TRUE`...
- The code acts like this:

```
void printit( char buf[] ) {
   int i=0;
   while ( TRUE ) {
      serialtx(buf[i] );
      i=i+1;
   }
}
...
```

- As a result, the serial data line continues to output serial data corresponding to the entire memory of the processor (program and data).
- Note that in small systems like this, memory is not protected, and `buf[0]...buf[65536]` is ALL of memory.

Protecting from this attack?

Consider this crazy code...

```
void printit( char buf[] ) {
   int i=0;
   while ( buf[i]!=0 AND buf[i]!=0 ) {
      serialtx(buf[i] );
      i=i+1;
   }
}
...
   printit( "Vsn 0.97" );
```

- Can you see why this code may be better than the previous code?
- Can you understand why a compiler might undermine this solution?

PK space attack

Smart cards are slow...

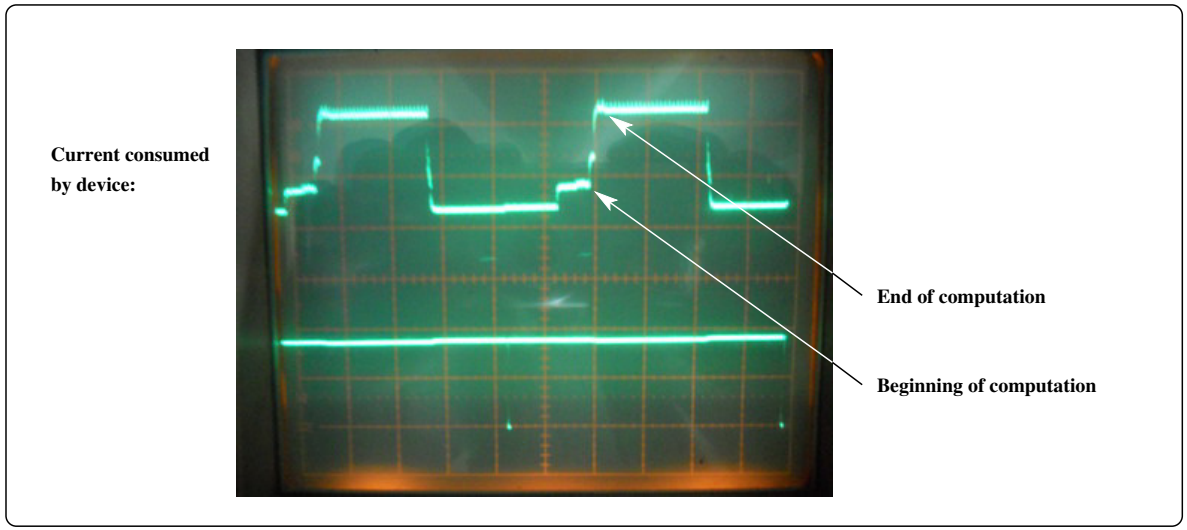
- Sometimes people implement PK encryption or authentication on smartcards, but the processors are really slow, and...
- An exponentiation operation required takes time proportional to the number of bits that are 1 in a large (1024) bit random prime number.
- A 1024 bit random number should on average have ..... bits set to a 1.
- However, this might make the smartcard unacceptably slow. So instead...
  - Developers select large (1024) bit prime numbers that only have a small number (3-5) bits set to a 1.
  - Smartcard is now fast, but open to attack.

What is the attack?

- Consider how few large (1024) bit primes there are, if only 5 bits are set to a 1!

Simple timing attack

Observe behaviour externally by measuring current...



- Current trace gives evidence about computation - type and timing.

What is the attack?

- Accurate measure of time of computation leads to an attack.

Simple timing attack...

A simple attack possible...

- The case of a function to test an 8-digit PIN number:

```
void testPIN( char buf[] ) {
   int i=0;
   while ( i<9 ) {
      if ( buf[i]!=password[i] ) {
         return NOMATCH;
      }
      i=i+1;
   }
   return MATCH;
}
```

- Instead of requiring $10^8 = 100,000,000$ attempts at the password, our attack algorithm only requires 80.
- The attack involves precise measurement of the time for the function while trying out PINs.

Simple timing attack...

Possible solution...

- A constant-time function to test an 8-digit PIN number:

```
void testPIN( char buf[] ) {
   int i=0;
   bool match=TRUE;
   while ( i<9 ) {
      if ( buf[i]!=password[i] ) {
         match=FALSE;
      } else {
         match=match;
      }
      i=i+1;
   }
   return match;
}
```

- No timing attack possible.
- Routine always takes a fixed amount of time.

Summary of security hardware, and attacks...

Only a brief outline here...

- Invasive and non-invasive attacks possible on security hardware.
- It is an arms race, and I have not even begun to cover techniques like encryption of memory, light sensitive fusing and so on.
- Invasive attacks may include
  - Decapsulation followed by imaging, reconnecting blown fuses, or probing.
  - Introducing faults, rewiring.
- Non-invasive attacks may include
  - Introducing glitches, or uncovering protocol/software flaws
  - Timing attacks.
- Software based techniques to reduce these effects include
  - Constant time loops
  - Reducing the likelihood of a single-glitch attack, by making code reliant on two independent tests.

Outline

Attack involves ECC, memory caches, timing...

OpenSSL: An important bit of security software

Why?
Because it is used in about 60% of all web based ssl (https) servers.
As a result of this we are very interested in bugs in openssl.
It may be the software most closely examined for security errors - all 700,000 lines of code!

What is the (one second) attack?

From the paper at http://eprint.iacr.org/2014/140.pdf .

> *"Our attack recovers the scalar k and thus the secret key of the signer and would therefore allow unlimited forgeries."*

The attack is a side channel attack. An attacker can determine if code has been cached or not. If it has been used, then it will be cached.

Timing attack on a constant time function!

Simple, and the Montgommery ladder:

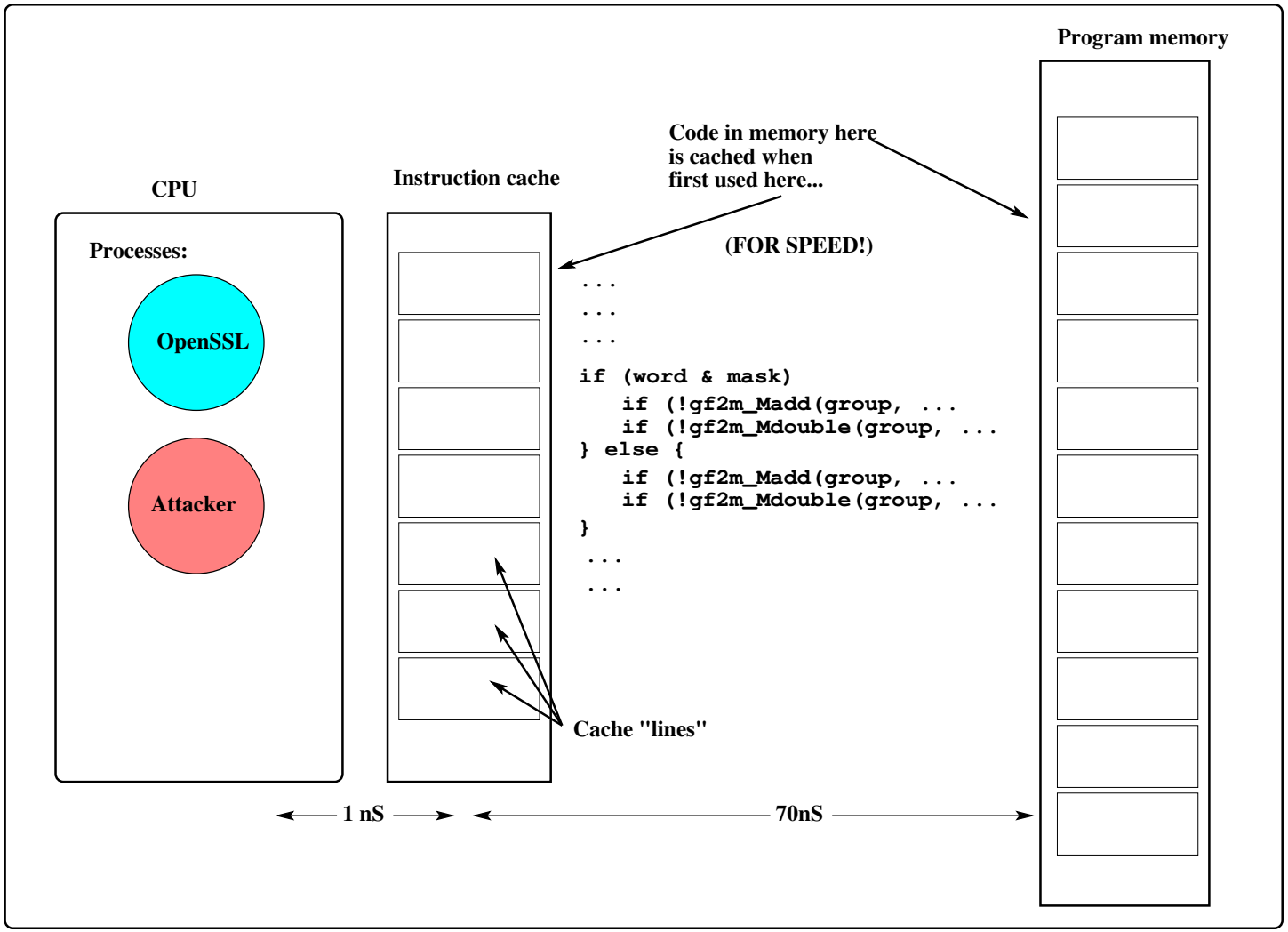
```
Q = O;
foreach bit in key {
   Q = 2Q;
   if (bit==0) {
      Q = Q + P;
   }
}
return Q;
```

```
R0 = O;
R1 = P;
foreach bit in key {
   if (bit==0) {
      R1 = R0 + R1;
      R0 = 2R0;
   } else {
      R0 = R0 + R1;
      R1 = 2R1;
   }
}
return R0;
```

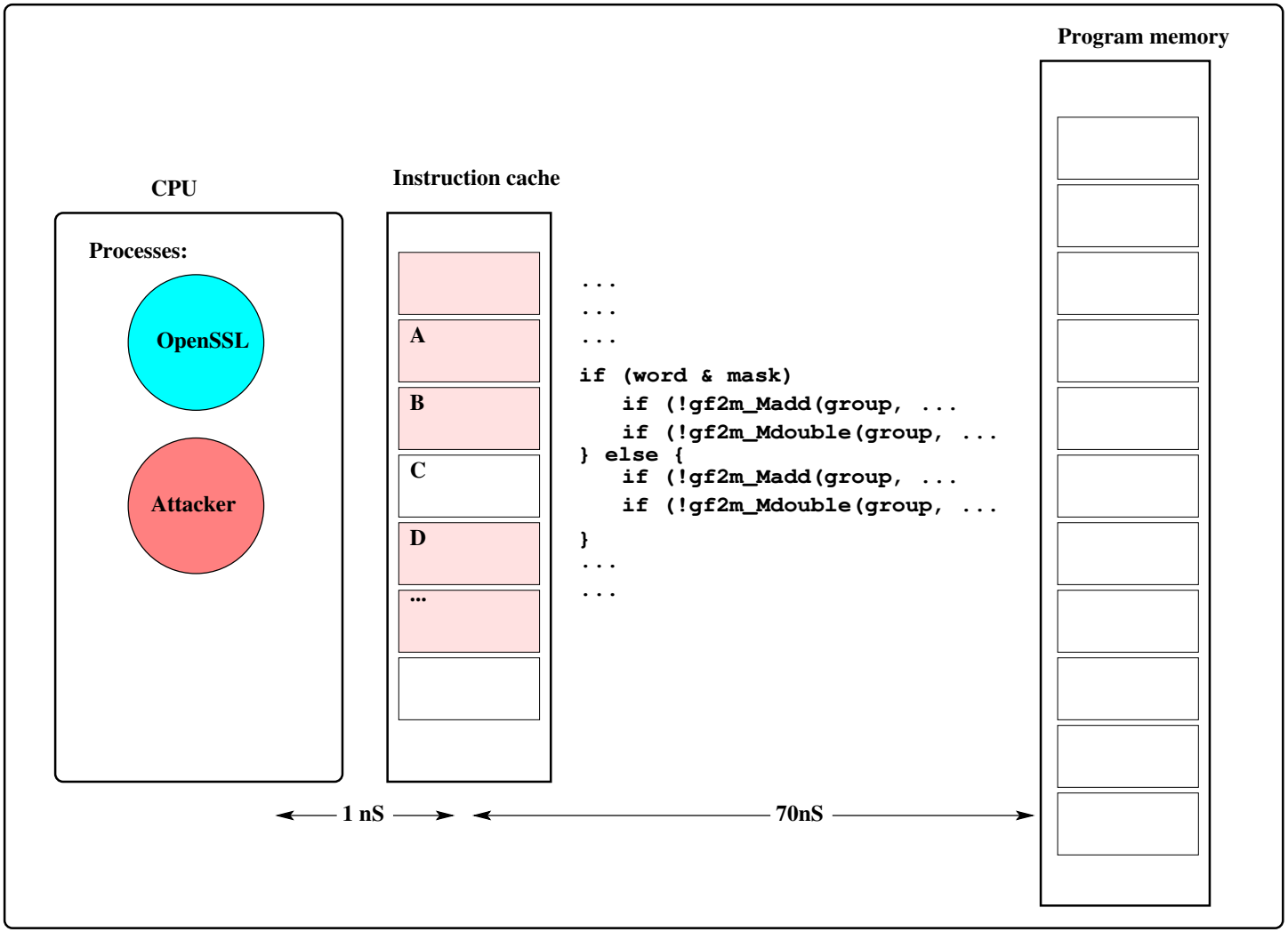
Both functions return the same result, but the one on the right runs in a constant time...
OpenSSL is built with security in mind, so (of course) it uses the Montgommery ladder - the code to the right, that runs in constant time.

Attack is directed at the instruction cache...

As openssl program runs, it caches the code...

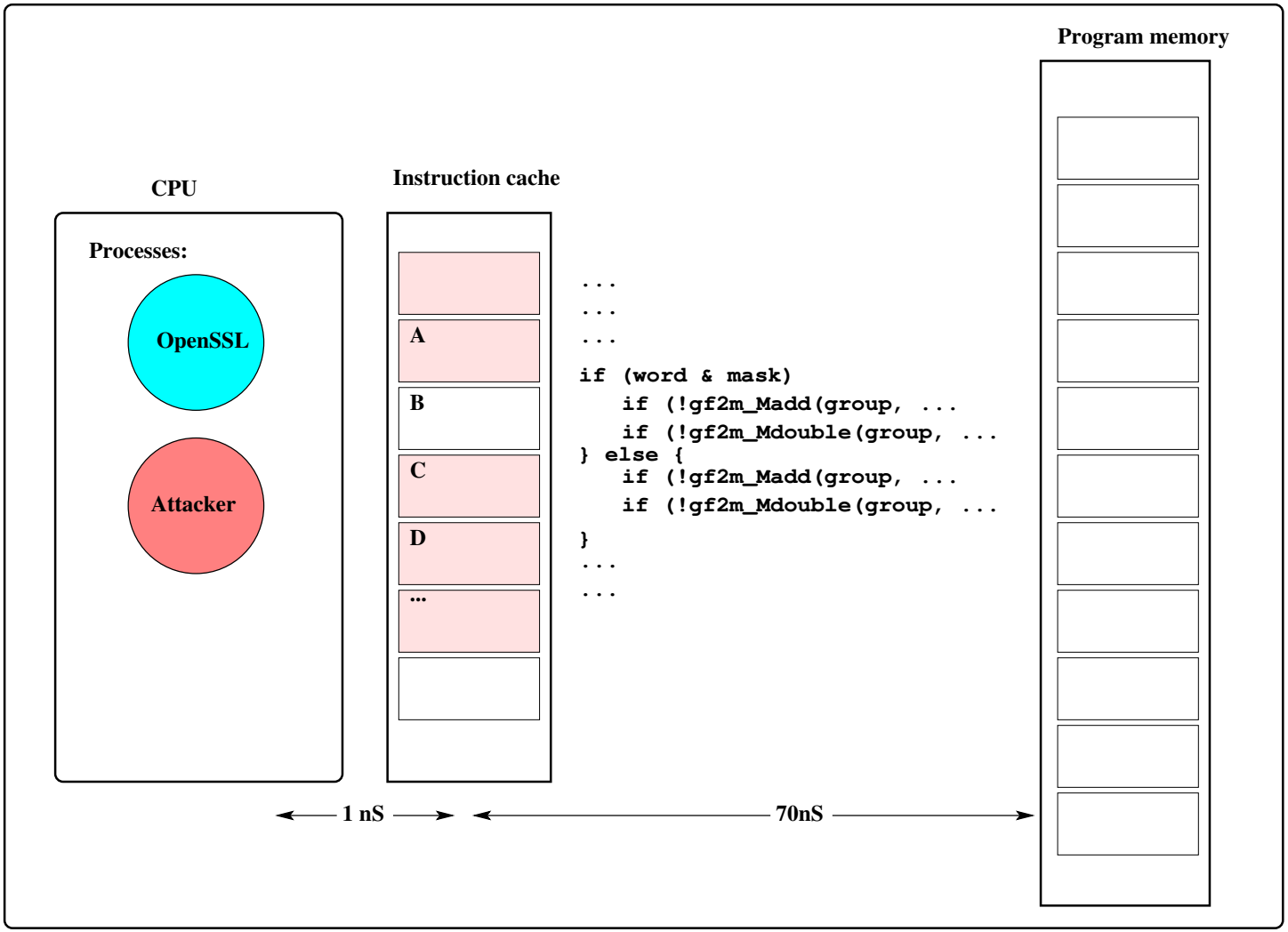


If bit is a “1” THIS is cached...



(and the attacker can determine this by seeing how fast the four cachelines A,B,C,D are loaded)

If bit is a “0” THIS is cached...



(and the attacker can determine this by seeing how fast the four cachelines A,B,C,D are loaded)

OpenSSL timing

Original openssl-1.0.1f/crypto/ec/ec2_mult.c...

```
if (word & mask) {
   gf2m_Madd(..., x1, ...
} else {
   gf2m_Madd(..., x2, ...
}
```

Each bit of a word is being tested, and depending on the bit we take two different paths. The paths operate on different (code) memory locations. If a particular path is taken, then a particular memory location would be cached, and an attacker can (afterwards) discover that, because that memory location would load faster. In this way an attacker can determine which path was taken and hence the value of each bit.

Fixed openssl-1.0.1g/crypto/ec/ec2_mult.c...

```
BN_consttime_swap(word & mask, x1, x2,...
gf2m_Madd(...
BN_consttime_swap(word & mask, x1, x2,...
```

We use a constant time swap, to put the variables in the same memory locations.

Outline

Summary...

Our surfaces...

1. People
2. Complexity
3. Cryptography
4. Communication media
5. IP and other networks
6. Web/app (and other) applications: their weak interfaces
7. Machines and their memory
8. Infrastructure, systems and mechanisms

Our defences...

The primary defence is a realistic awareness of the level of threat around you, and a proactive approach to securing your systems.
I do hope this course gave you some ideas...

End statement...

It has been a privilege to have taught you, and I hope I have shown you a few things that are interesting to you, and help you in the future...