# ECEN 5053-002

## Developing the Industrial Internet of Things

Week 5 - Lecture

Security

Dave Sluiter - Spring 2018

# Warning - Ethics, Law, and University Policies

To defend a system you need to be able to think like an attacker, and that includes understanding techniques that can be used to compromise security. However, using those techniques in the real world may violate the law or the University's rules, and it may be unethical. Under some circumstances, even probing for weaknesses may result in severe penalties, up to and including expulsion, civil fines, and jail time. Our policy in this class is that you must respect the privacy and property rights of others at all times, **or else you will fail the course**.

Acting lawfully and ethically is your responsibility. Carefully read the Computer Fraud and Abuse Act (CFAA), a federal statute that broadly criminalizes computer intrusion. This is one of several laws that govern "hacking." Understand what the law prohibits — you don't want to end up like this guy. If in doubt, we can refer you to an attorney.

Please review CU's acceptable use policy of IT resources for guidelines concerning proper use of information technology, as well as the Engineering Honor Code.

Source: Eric Wustrow

# Material

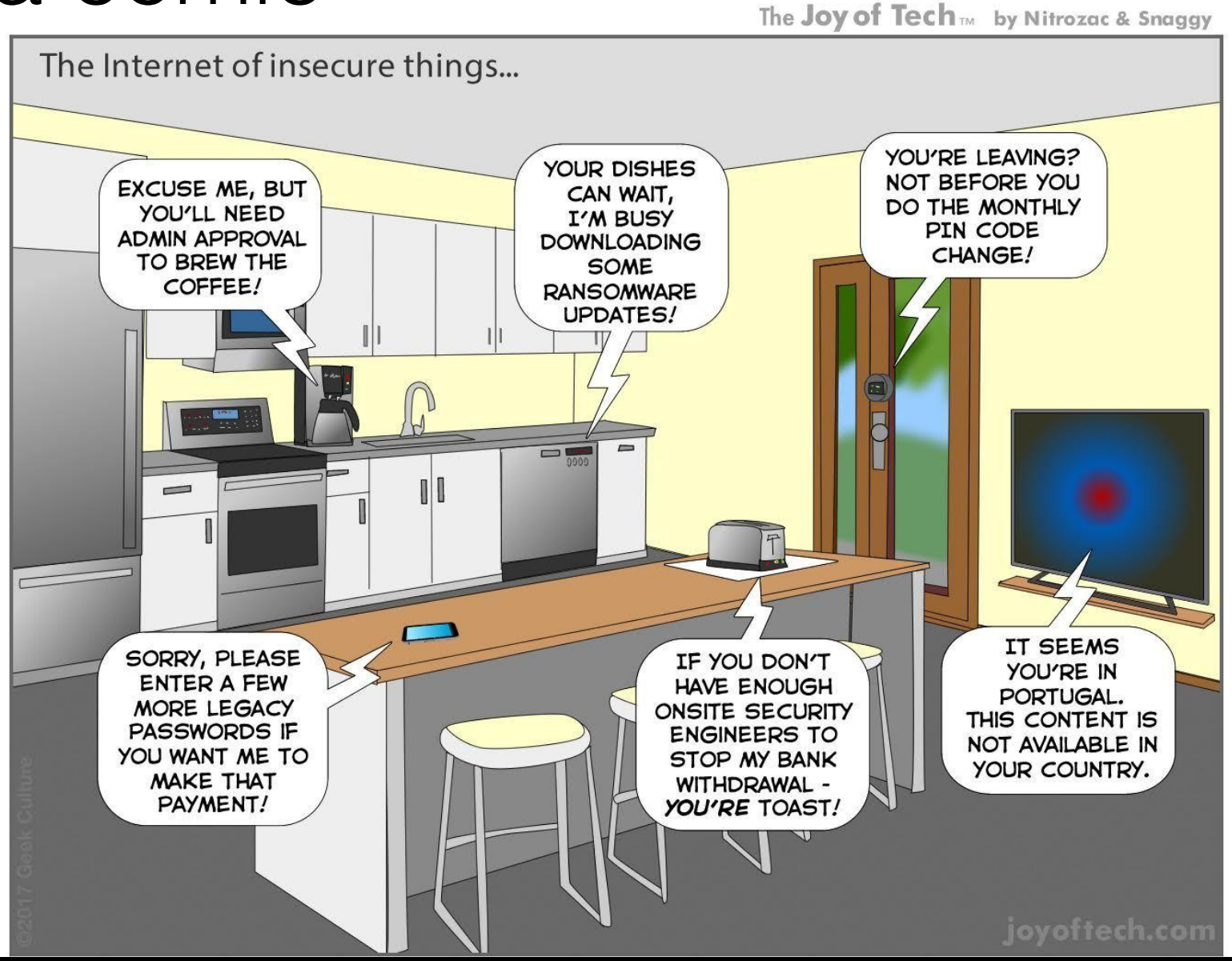
- Security overview
- Encryption techniques, one time pad, symmetric encryption, asymmetric encryption
- Diffie-Hellman
- Hashes, Message Authentication Codes (MACs)
- Attack vectors: AES, key, man-in-the-middle, replay
- Key protection, hardware techniques
- Side-channel attacks
- Chain of Trust
- Examples of poor security implementations
- How web browsers establish a secure connection
- Blockchains

# Learning Outcomes

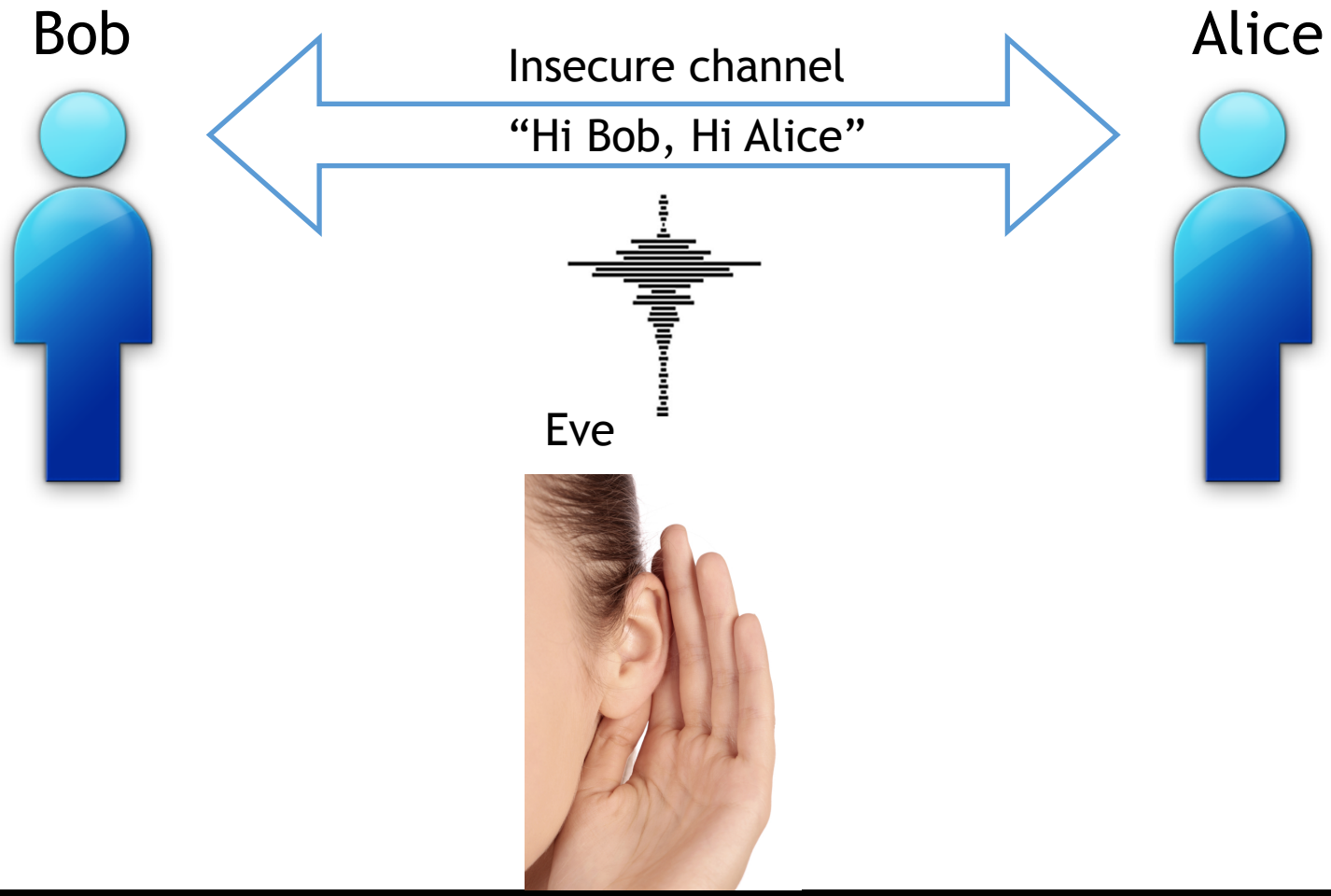
- Develop a “Security” mindset
- Address security at all levels and at all interfaces in a system
  - Make it difficult for attackers to walk through the door.
- Difference between symmetric and asymmetric encryption, Diffie-Hellman, Hashes, MACs, key protection schemes, man-in-the-middle and replay attacks
- Awareness of US security standards

# Security Overview

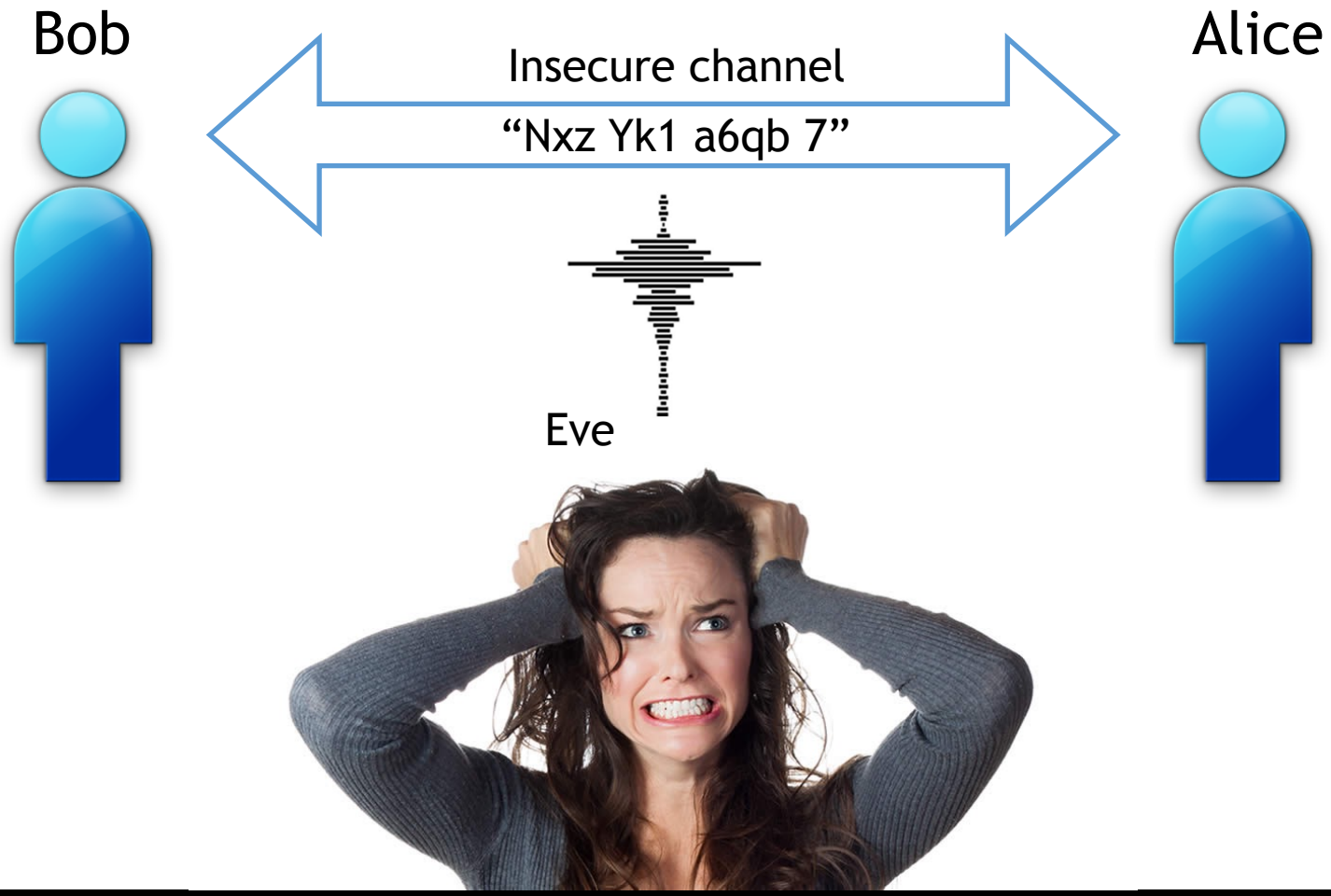
# But first, a comic



# What does it mean to be secure?



# What does it mean to be secure?



# The Security Mindset - Bruce Schneier

- https://www.schneier.com/blog/archives/2008/03/the_security_mi_1.html

“Security requires a particular mindset. Security professionals -- at least the good ones -- see the world differently. They can't walk into a store without noticing how they might shoplift. They can't use a computer without wondering about the security vulnerabilities. They can't vote without trying to figure out how to vote twice. They just can't help it.

SmartWater is a liquid with a unique identifier linked to a particular owner. "The idea is for me to paint this stuff on my valuables as proof of ownership," I wrote when I first learned about the idea. "I think a better idea would be for me to paint it on *your* valuables, and then call the police.” Really, we can't help it.

This kind of thinking is not natural for most people. It's not natural for engineers. Good engineering involves thinking about how things can be made to work; **the security mindset involves thinking about how things can be made to fail**. It involves thinking like an attacker, an adversary or a criminal. You don't have to exploit the vulnerabilities you find, but if you don't see the world that way, you'll never notice most security problems.”

# The Security Mindset - Dave Sluiter

- When working in security, it is an unwise mental mindset to make statements such as: “That’s impossible”, or “No one will ever figure this out” and other such absolute statements.
- A better mindset is one that blurs the line between TRUE and FALSE, mental positions such as likely/unlikely, probable/improbable, practical/impractical.
- The world is full of some very clever and well funded people.
    - Examples:
        - WWII German Enigma machine
        - US NSA
        - Israeli Mossad
- There is no 100% security, only approaches/solutions deemed “good enough”.
- “Security through obscurity is not security” - courtesy of Don Matthews

# Kerckhoffs's Principle

- Stated by Dutch cryptographer Auguste Kerckhoff in the 19th century:
  - A crypto system should be secure even if everything about the system, **except the key**, is public knowledge.

# The Security Mindset

- What can we learn from wood-block puzzles?
- How to think “orthogonal” or unconventional

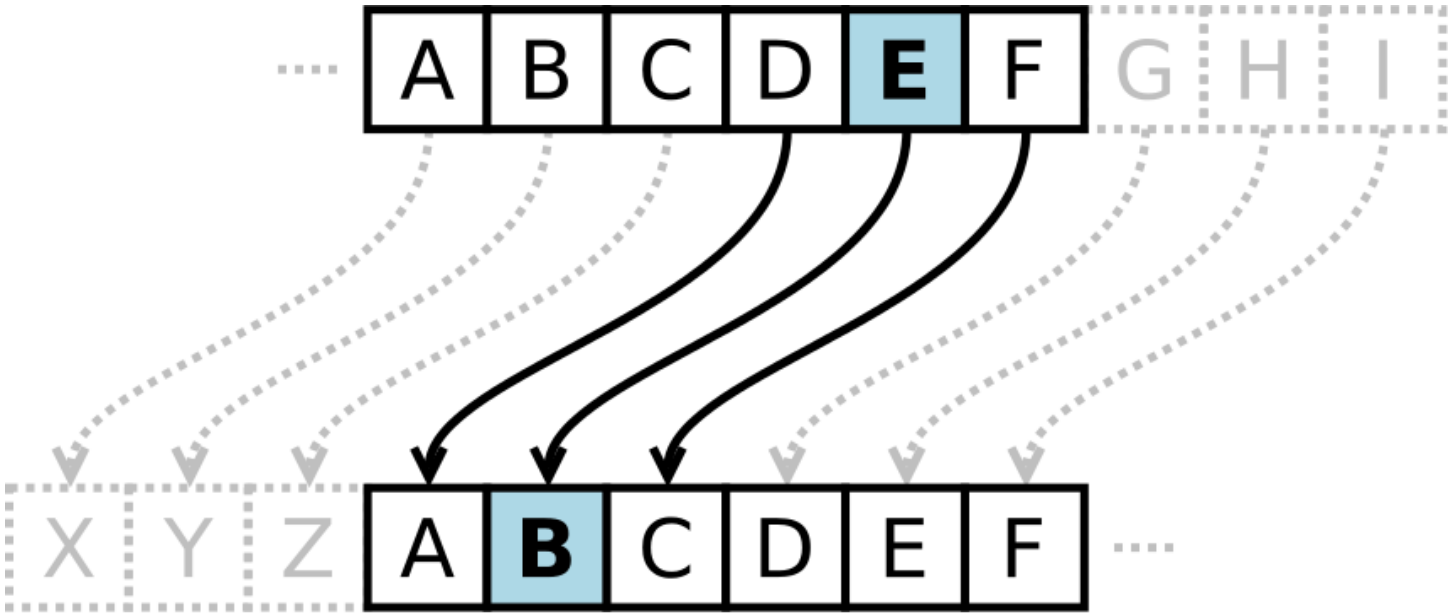
# Basics

# Terminology

- **Plaintext**, often denoted by P (sometimes called clear-text), is what you want to protect/encrypt
- **Ciphertext**, often denoted by C, is what you get after encryption

# Encryption Techniques

- Caesar Cipher, named after Julius Caesar
- Substitution cipher, based on a shift



# Encryption Techniques

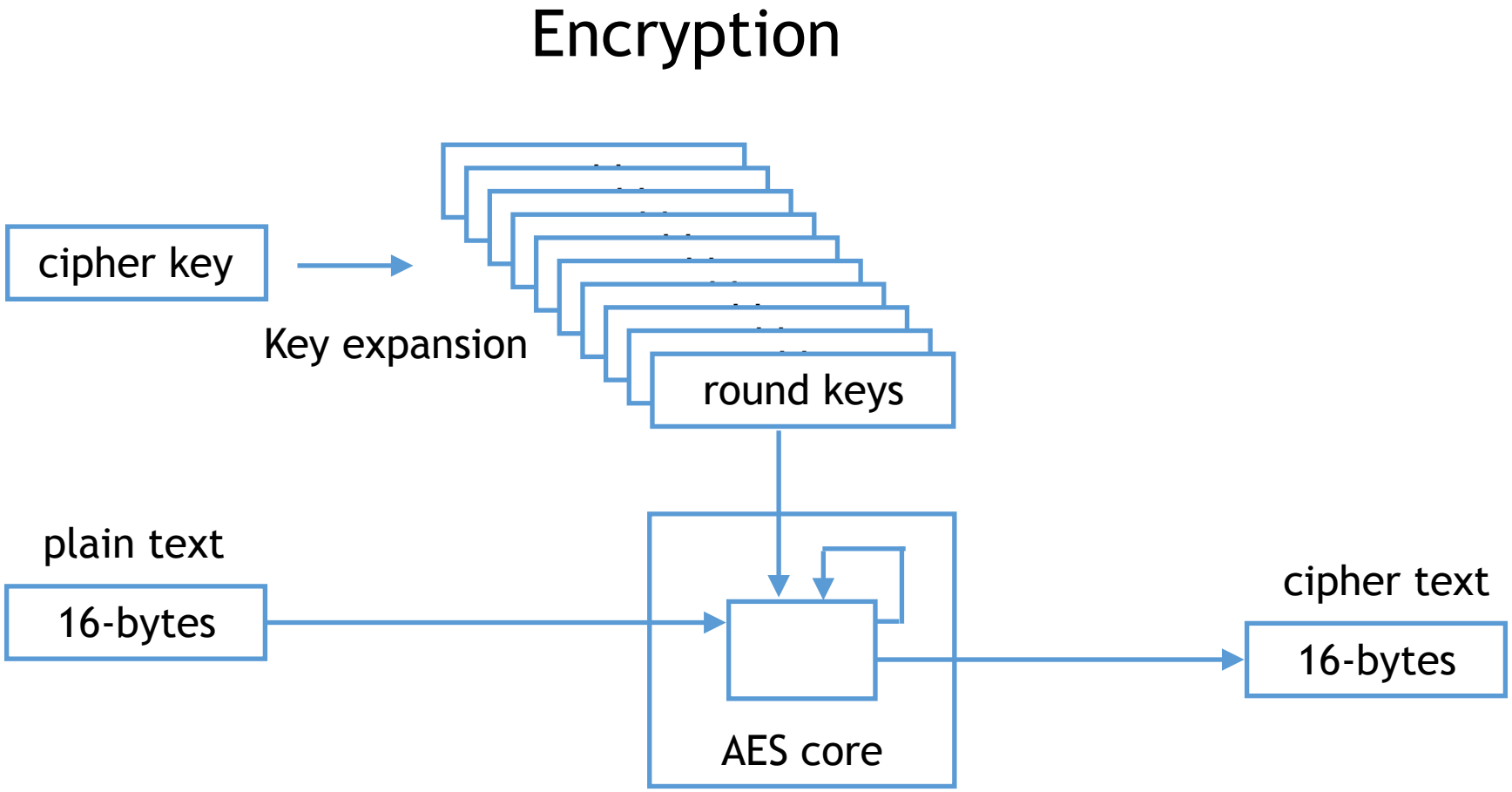
- One time pad (OTP)
- So-called “perfect” encryption
- Impractical for real-world

| A=0, B=1, C=2, … Z=25 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Plain text** | M | E | E | T | T | O | N | I | G | H | T |
| | 12 | 4 | 4 | 19 | 19 | 14 | 13 | 8 | 6 | 7 | 19 |
| | | | | | | | | | | | |
| **Key** | D | Z | H | S | U | I | M | W | E | K | C |
| | 3 | 25 | 7 | 18 | 20 | 8 | 12 | 22 | 4 | 10 | 2 |
| | | | | | | | | | | | |
| **Sum** | 15 | 29 | 11 | 37 | 39 | 22 | 25 | 30 | 10 | 17 | 21 |
| **Sum mod 26** | 15 | 3 | 11 | 11 | 13 | 22 | 25 | 4 | 10 | 17 | 21 |
| **Cipher Text** | P | D | L | L | N | W | Z | E | K | R | V |
| | | | | | | | | | | | |
| | | | | | | | | | | | |
| **Cipher text** | P | D | L | L | N | W | Z | C | K | R | V |
| | 15 | 3 | 11 | 11 | 13 | 22 | 25 | 4 | 10 | 17 | 21 |
| | | | | | | | | | | | |
| **Key** | D | Z | H | S | U | I | M | W | E | K | C |
| | 3 | 25 | 7 | 18 | 20 | 8 | 12 | 22 | 4 | 10 | 2 |
| | | | | | | | | | | | |
| **Diff** | 12 | -22 | 4 | -7 | -7 | 14 | 13 | -18 | 6 | 7 | 19 |
| **Sum mod 26** | 12 | 4 | 4 | 19 | 19 | 14 | 13 | 8 | 6 | 7 | 19 |
| **Plain text** | M | E | E | T | T | O | N | I | G | H | T |

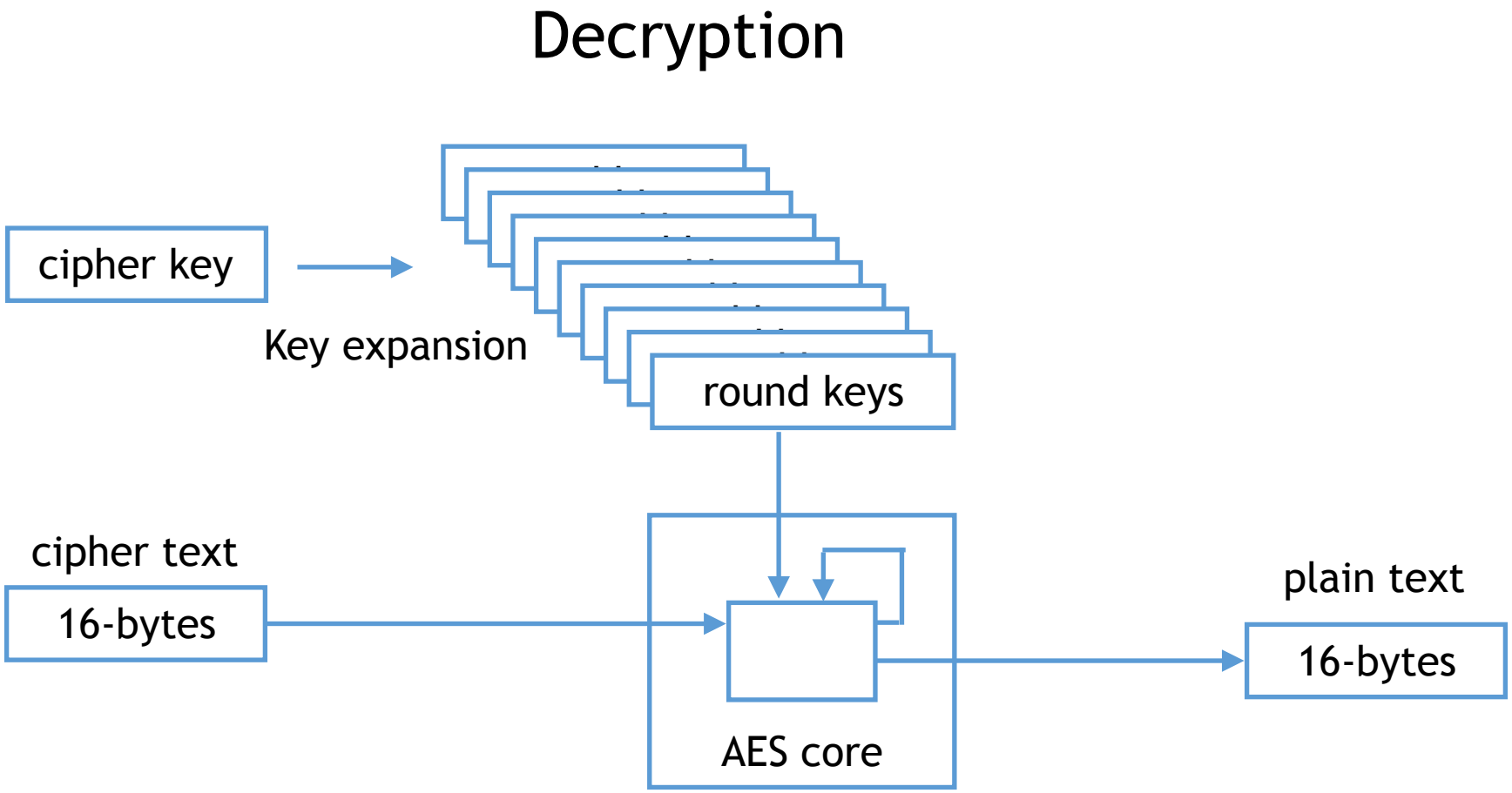
# AES (Advanced Encryption Standard)

- Established by US NIST (National Institute of Standards)
- Block cipher: 16-bytes in, 16-byte out
- 3 key lengths: 128-, 192-, 256-bits
- High-level description
  - 1) With N=number of rounds, round keys are extracted from the cipher key (where N = 10, 12, 14, for 128-, 192- or 256-bits)
  - 2) Round 0
  - 3) Rounds 1 to N-2
  - 4) Final round N-1
- Believed to be secure, but we don't know how to prove it

# AES Encryption



# AES Decryption

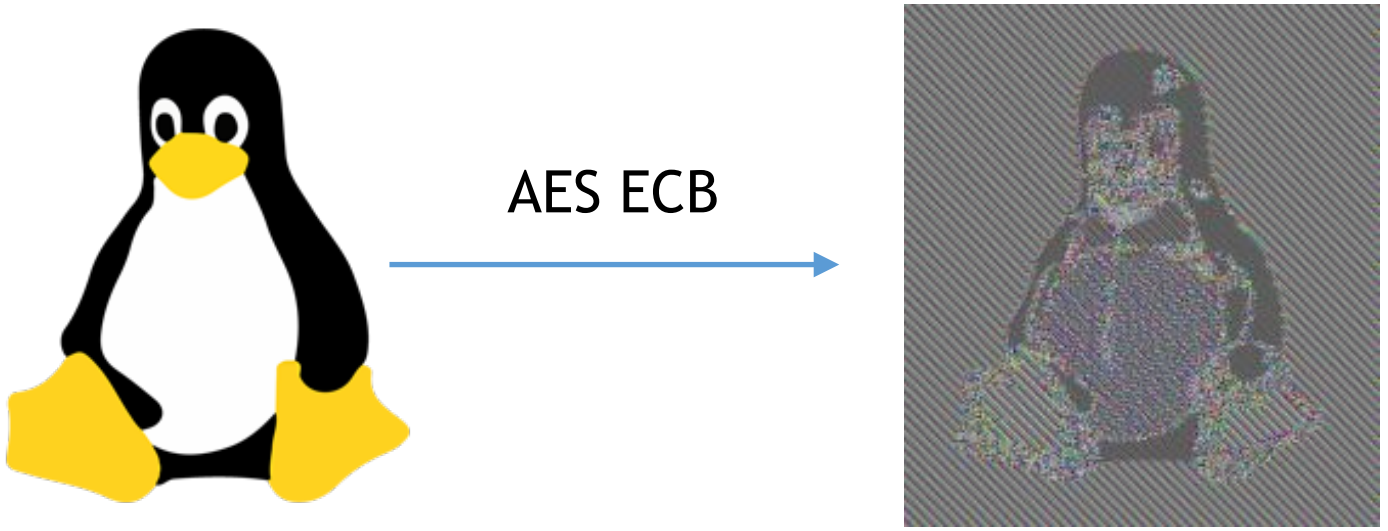


# AES

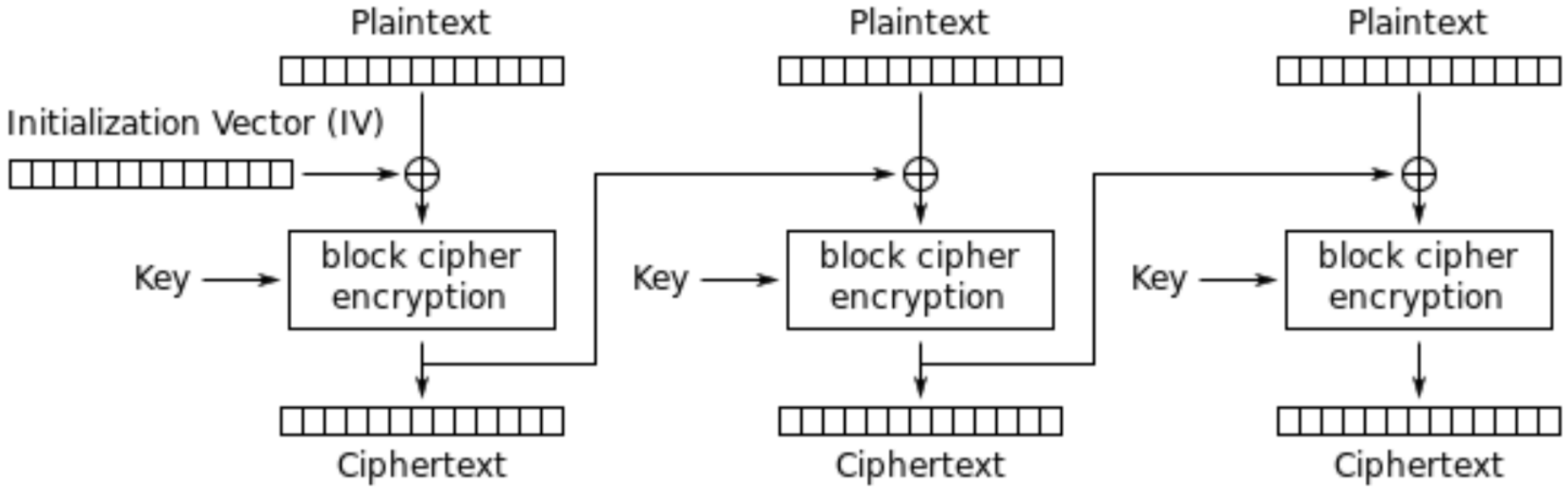
- When AES is implemented as per specification, it is known as Electronic Code Book (ECB)
- Shouldn’t this be good enough?
- Can you think of any issues that might arise?

# AES ECB mode

- The same plain-text always encrypts to the same cipher-text
- The result is, it leaks information

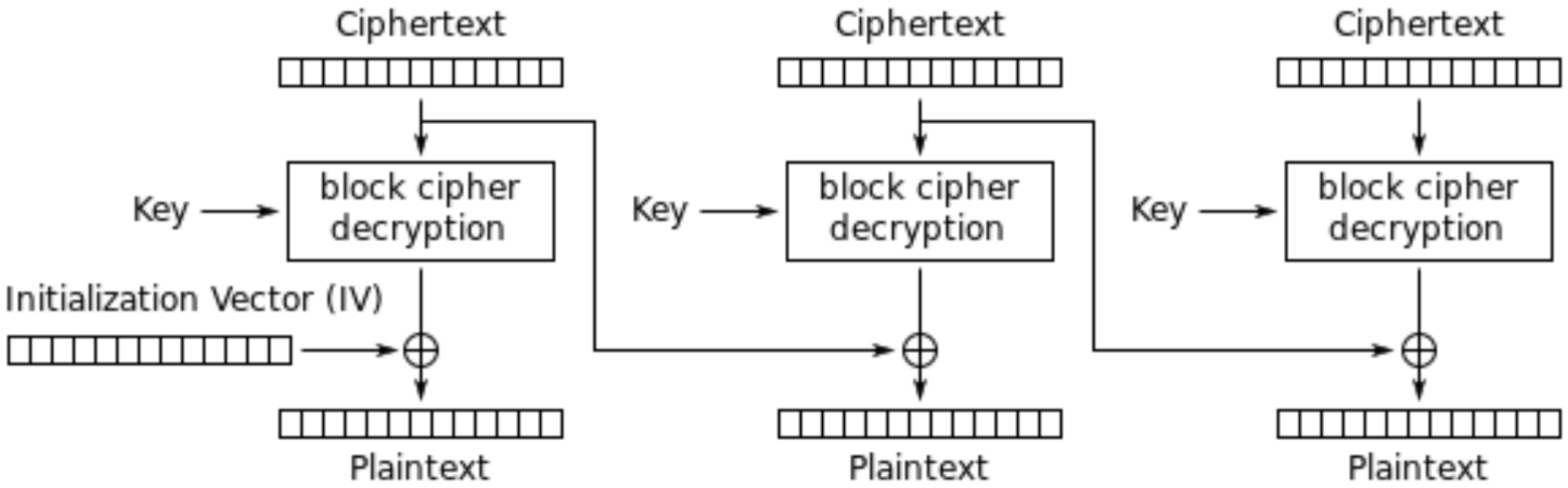


# AES CBC (Cipher Block Chaining Mode)



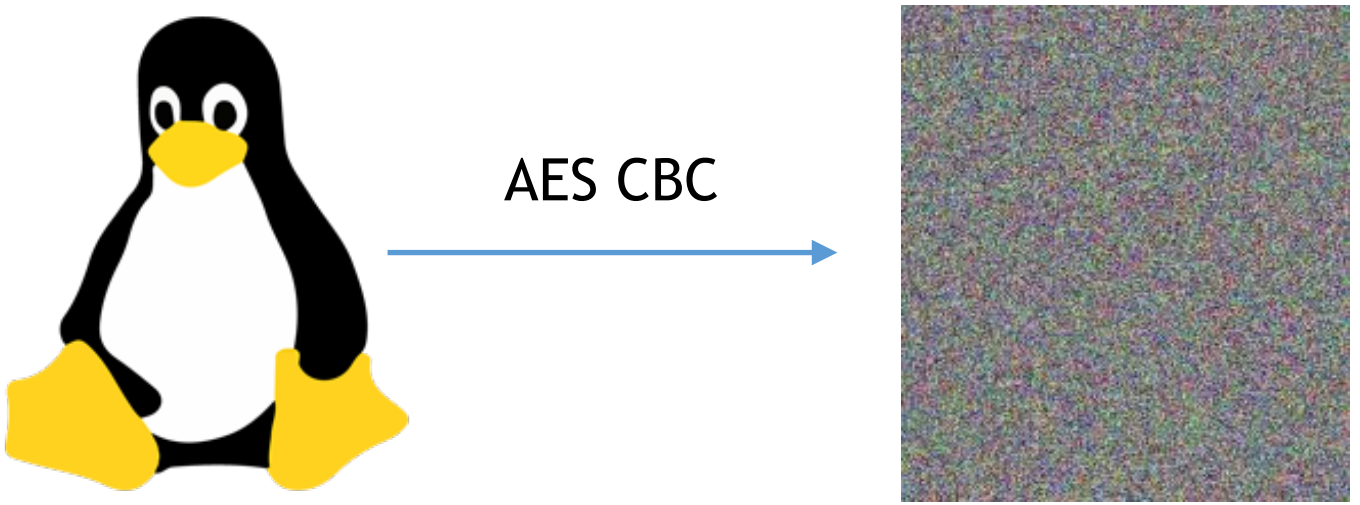

Cipher Block Chaining (CBC) mode encryption

# AES CBC (Cipher Block Chaining Mode)




Cipher Block Chaining (CBC) mode decryption

# AES CBC mode



# AES CBC mode

- Initialization vector/value (IV) should be chosen wisely
- Ideally, each 16-byte block would have a unique AND unpredictable IV (not a counter)

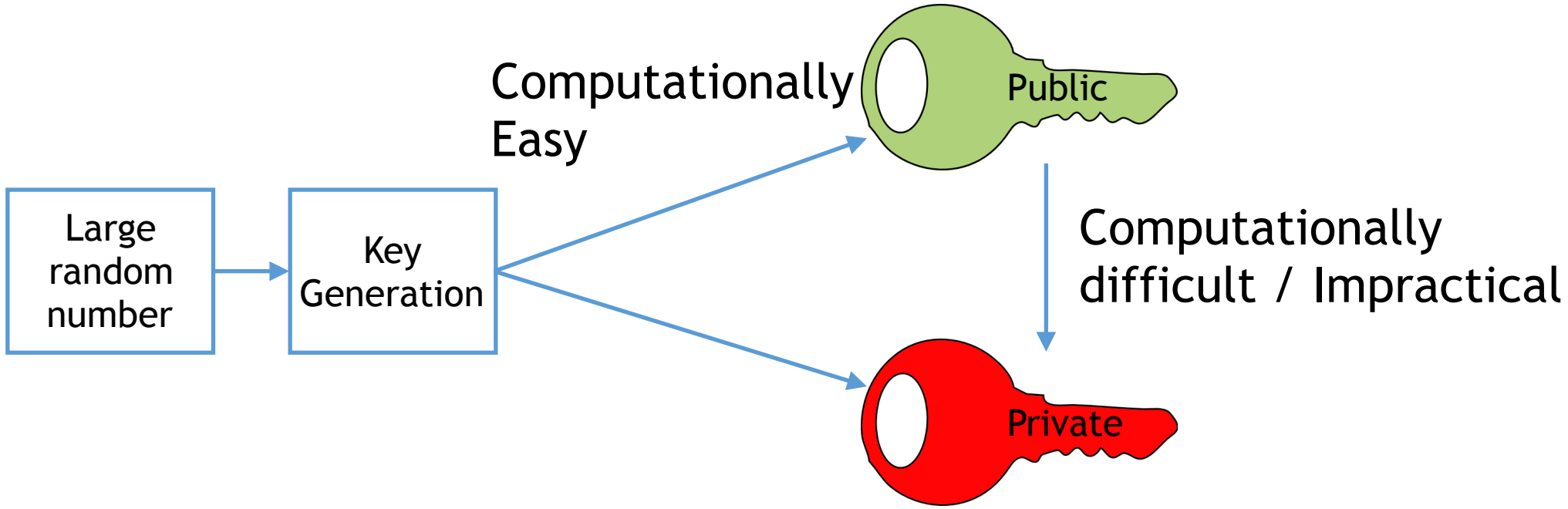
# AES XTS mode

- NIST added XTS mode for storage devices
- Specified in SP800-38E
- Deemed “better than” CBC
- AES is known as a **symmetric** encryption algorithm because the same key is used for encryption and decryption

# Asymmetric Encryption

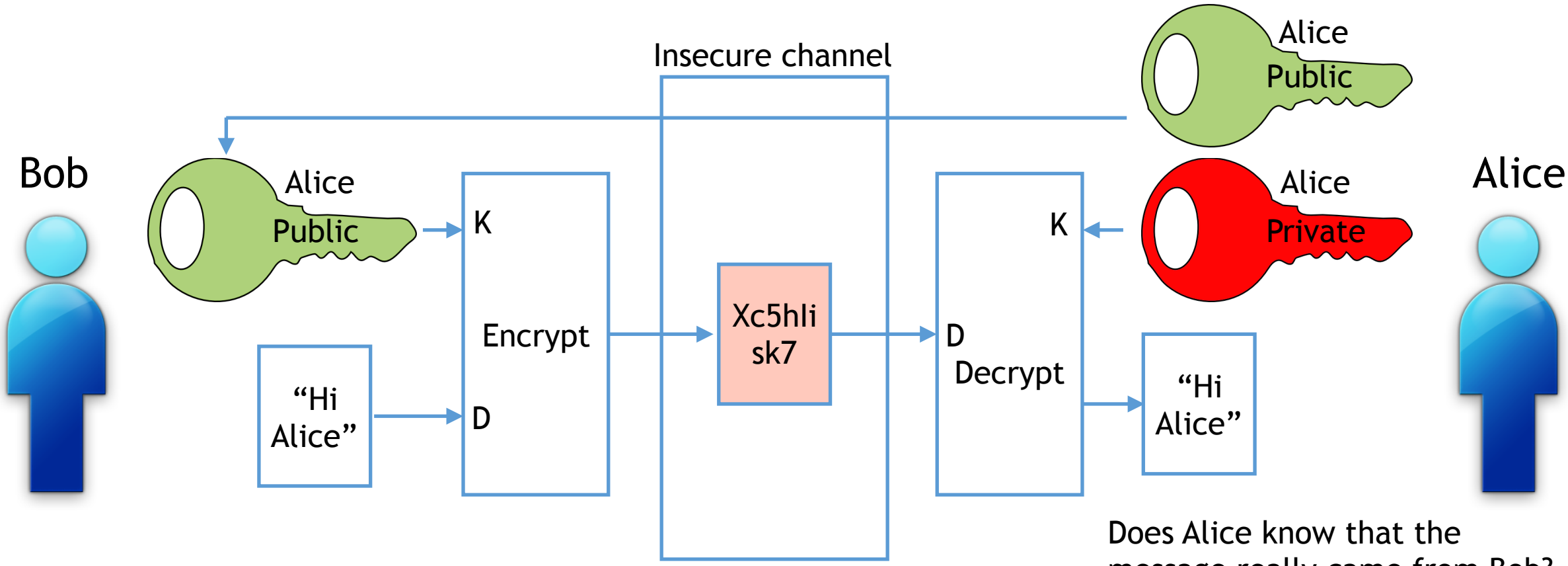
- Also known as Public Key Encryption (Public Key Cryptography)
- Uses a pair of keys: 1 public, 1 private
- Provides two functions:
  - Encryption
  - Authentication, verifies the message came from the holder of the matching private key

# Asymmetric Encryption



# Asymmetric Encryption

Alice wants to receive secure communication from Bob

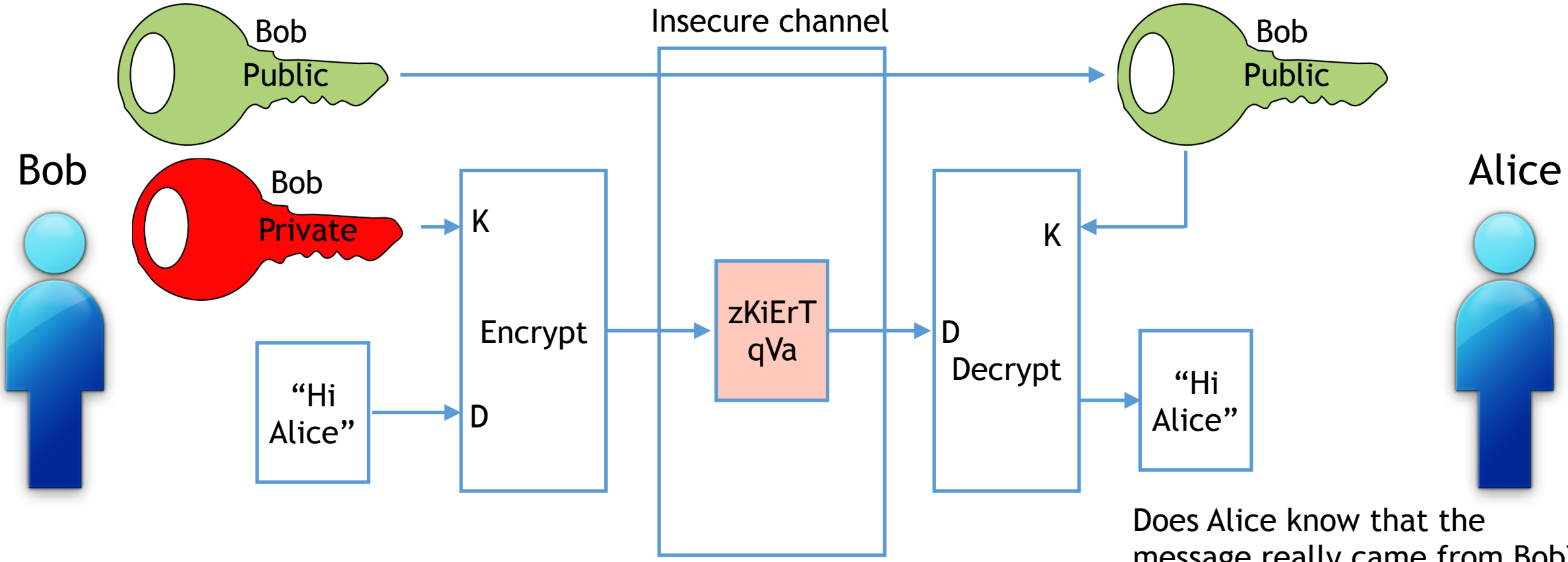



Does Alice know that the message really came from Bob?

What if **Eve** had Alices's public key?

# Asymmetric Encryption

Alice wants to receive secure communication from Bob

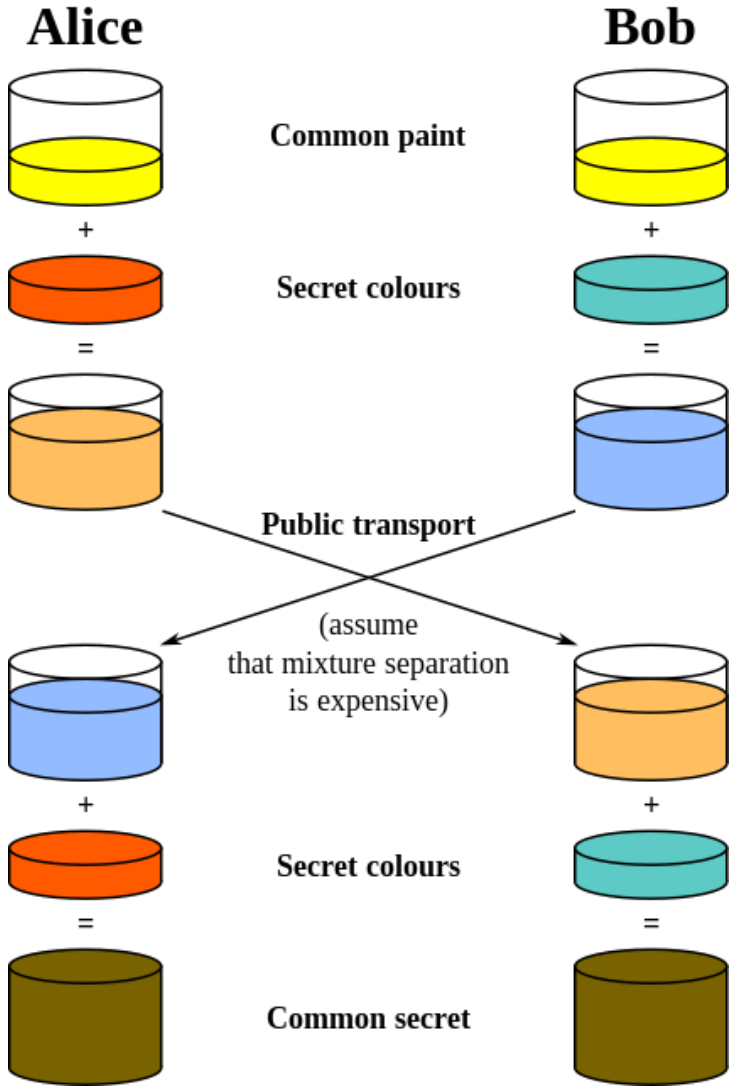



Does Alice know that the message really came from Bob?

What if **Eve** had Bob's public key?

# Diffie-Hellman

A method to securely establish a known secret (a “key”) between 2 parties over an insecure channel.




Source: https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange

# Diffie-Hellman

Note: Larger values of a, b and p would be needed to make this secure, g is usually a small number.

Also Note:

$$(g^a)^b = (g^b)^a$$

The simplest and the original implementation of the protocol uses the multiplicative group of integers modulo $p$, where $p$ is prime, and $g$ is a primitive root modulo $p$. These two values are chosen in this way to ensure that the resulting shared secret can take on any value from 1 to $p$–1. Here is an example of the protocol, with non-secret values in blue, and secret values in **red**.

1. Alice and Bob agree to use a modulus $p = 23$ and base $g = 5$ (which is a primitive root modulo 23).
2. Alice chooses a secret integer $\mathbf{a} = 6$, then sends Bob $A = g^a \bmod p$
   - $A = 5^6 \bmod 23 = 8$
3. Bob chooses a secret integer $\mathbf{b} = 15$, then sends Alice $B = g^b \bmod p$
   - $B = 5^{15} \bmod 23 = 19$
4. Alice computes $\mathbf{s} = B^a \bmod p$
   - $\mathbf{s} = 19^6 \bmod 23 = \mathbf{2}$
5. Bob computes $\mathbf{s} = A^b \bmod p$
   - $\mathbf{s} = 8^{15} \bmod 23 = \mathbf{2}$
6. Alice and Bob now share a secret (the number **2**).

Source: https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange
Primitive root mod n: http://math.stackexchange.com/questions/795414/what-are-primitive-roots-modulo-n

# Diffie-Hellman

| Alice | | Bob | | Eve | |
|---|---|---|---|---|---|
| **Known** | **Unknown** | **Known** | **Unknown** | **Known** | **Unknown** |
| $p = 23$ | | $p = 23$ | | $p = 23$ | |
| $g = 5$ | | $g = 5$ | | $g = 5$ | |
| $\boldsymbol{a = 6}$ | $\boldsymbol{b}$ | $\boldsymbol{b = 15}$ | $\boldsymbol{a}$ | | $\boldsymbol{a}$, $\boldsymbol{b}$ |
| $A = 5^a \bmod 23$ | | $B = 5^b \bmod 23$ | | | |
| $A = 5^6 \bmod 23 = 8$ | | $B = 5^{15} \bmod 23 = 19$ | | | |
| | | | | $A = 8$, $B = 19$ | |
| $\boldsymbol{B = 19}$ | | $\boldsymbol{A = 8}$ | | | |
| $\boldsymbol{s} = B^a \bmod 23$ | | $\boldsymbol{s} = A^b \bmod 23$ | | | |
| $\boldsymbol{s} = 19^6 \bmod 23 = \boldsymbol{2}$ | | $\boldsymbol{s} = 8^{15} \bmod 23 = \boldsymbol{2}$ | | | $\boldsymbol{s}$ |

Source: https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange

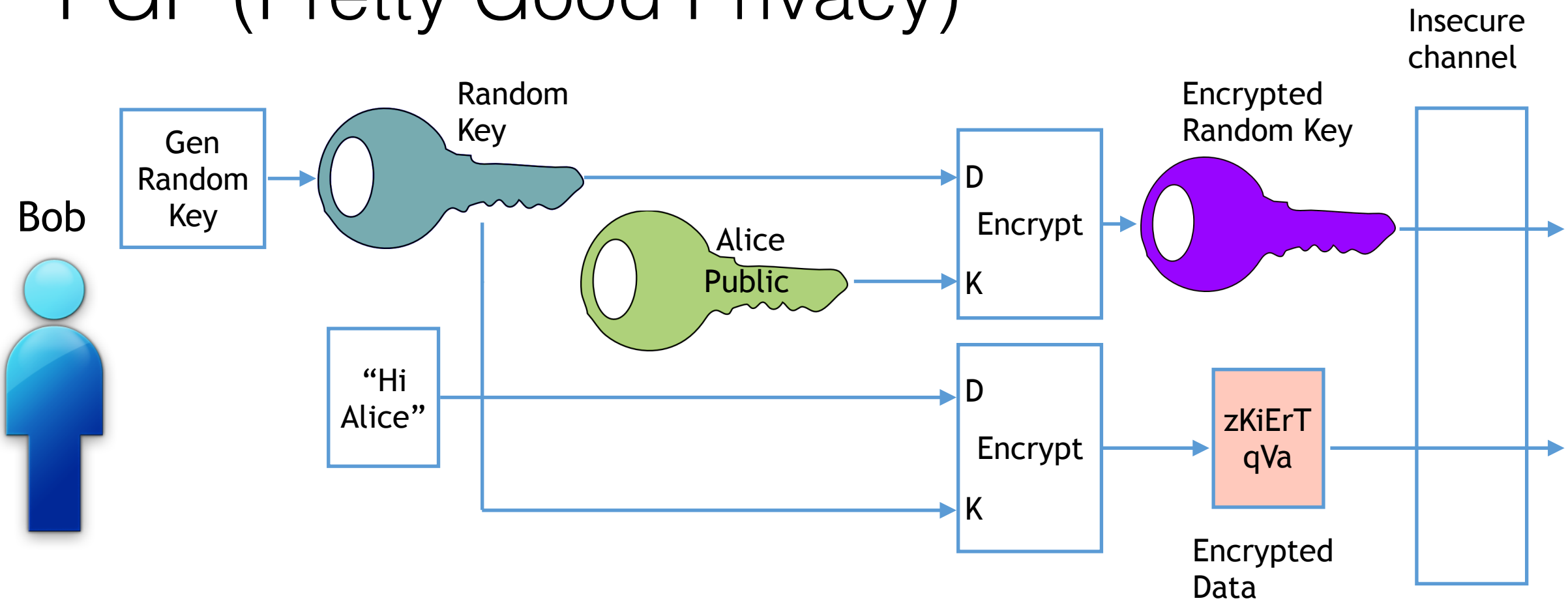
# Diffie-Hellman

- Alice and Bob now use symmetric encryption (AES XTS for example) on an insecure channel, using $s$ as the common encryption/decryption key.

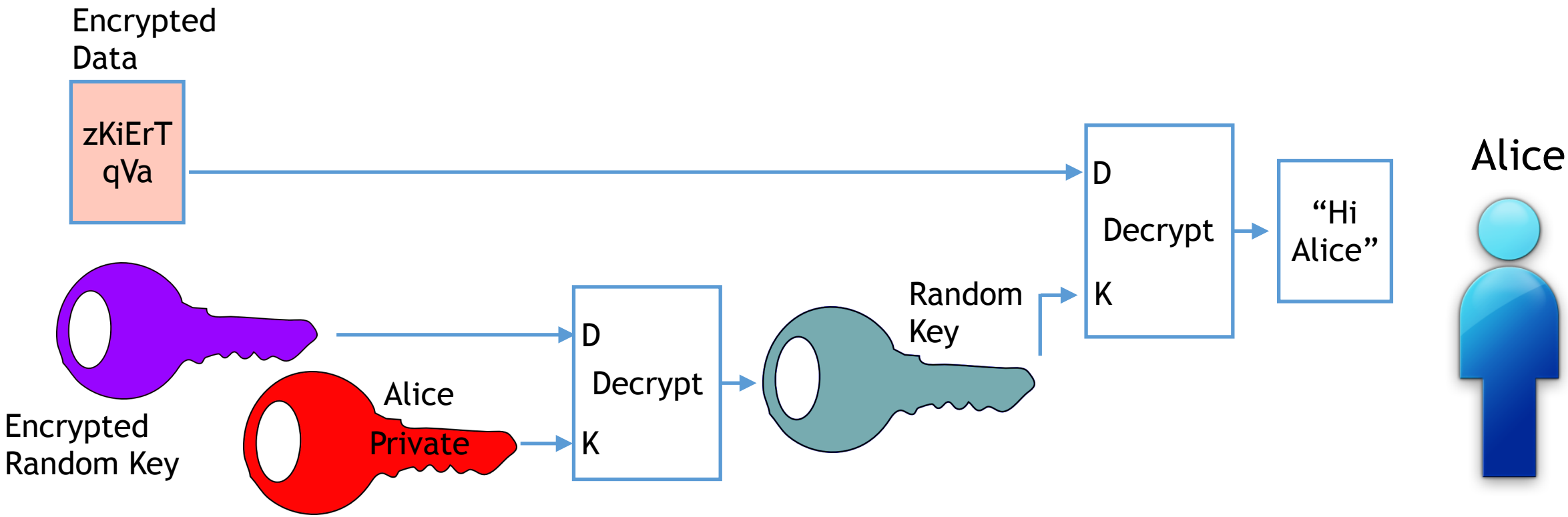
# PGP (Pretty Good Privacy)

- Created by Phil Zimmerman in 1991
- Follows the OpenPGP standard (RFC 4880)

# PGP (Pretty Good Privacy)



# PGP (Pretty Good Privacy)



# RSA (Rivest-Shamir-Adleman)

- Involves 4 steps
  - Key generation
  - Key distribution
  - Encryption
  - Decryption

Source: https://en.wikipedia.org/wiki/RSA_(cryptosystem)

# RSA

- It is practical to find 3 very large positive integers $e$, $d$ and $n$ such that with modular exponentiation for all integers $m$, we have:

$$(m^e)^d \equiv m(\mathrm{mod}\, n)$$

and

$$(m^d)^e \equiv m(\mathrm{mod}\, n)$$

$b^y \,\%\, m$

Integer $b$ raised to $y$, divided by the modulus $m$ to produce a remainder

And that even knowing $e$, $m$ and $n$, it can be very difficult to determine $d$

# RSA Key Generation

- Choose two distinct prime number $p$ and $q$
  - Chosen at random, similar magnitudes
- Compute $n = p*q$
- $n$ is used as the modulus, this is the key length
- Compute $\lambda(n) = lcm(p-1, q-1)$ kept private
- Choose an integer $e$, such that $1 < e < \lambda(n)$
  - where $e$ and $\lambda(n)$ are coprime
- Determine $d$ as $d \equiv e^{-1} (\mathrm{mod}\, \lambda(n))$

# RSA Key Distribution

- The public key consists of $n$ and $e$ and is distributed
- The private key consists of $n$ and $d$ and is kept secret
  - $p$, $q$ and $\lambda(n)$ must also be kept secret

# RSA Encryption

- Bob wants to send Alice a message M after receiving $n$ and $e$ from Alice
- Bob reduces his message to an integer $m$ where 0 <= $m$ <= $n$ by using a previously agreed upon reversible protocol known as a *padding scheme*. Cipher text is then computed as:

$$c \equiv m^e \pmod{n}$$

For padding explanation, see: https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding

# RSA Decryption

- Alice computes:

$$c^d \equiv (m^e)^d \equiv m (\mathrm{mod}\, n)$$

- And then reversing the padding scheme

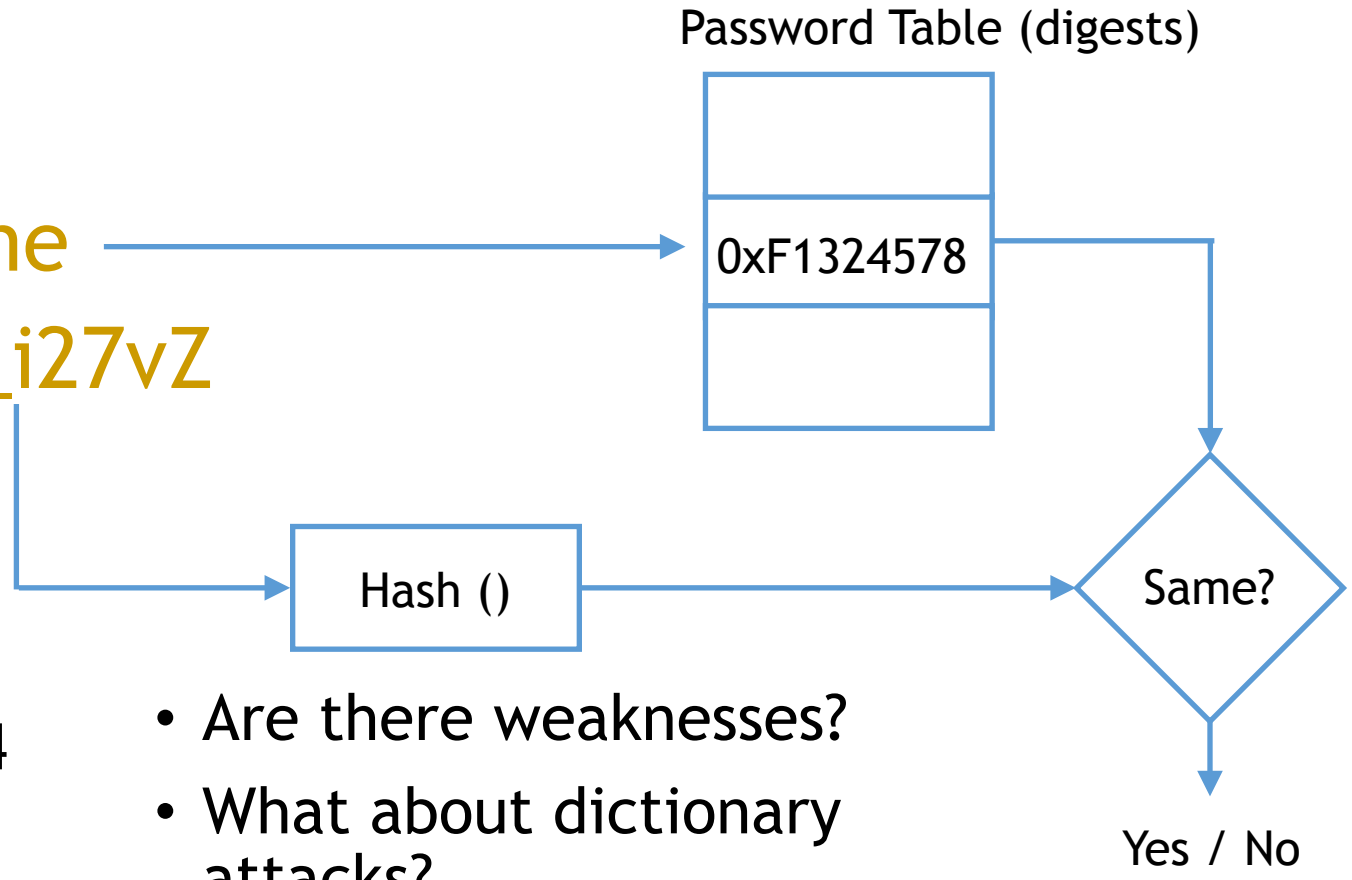
# Hash Functions

- Maps a message of arbitrary length to a fixed number of bits, referred to as the hash value (or message digest, or simply **digest**)
- "Good" cryptographic hash functions have 5 properties:
  - Deterministic: same input, same output every time
  - Quick to compute, computationally inexpensive
  - ***Impractical to determine a message from the hash value***
    - ***Has the notion of being a "one-way" function***
  - A small change in input results in a large change in the output
  - Impractical to find two messages that map to the same hash value
- Used to check the integrity of data transmission

See: https://en.wikipedia.org/wiki/Comparison_of_cryptographic_hash_functions for a list of cryptographic hash functions

# Uses for Hash Functions

- Saving passwords
  - Login: UserName
  - Password: Xh8_i27vZ




- See also the SHA-2 family of hash functions as per FIPS 180-4
- Well studied, haven't spotted a problem yet

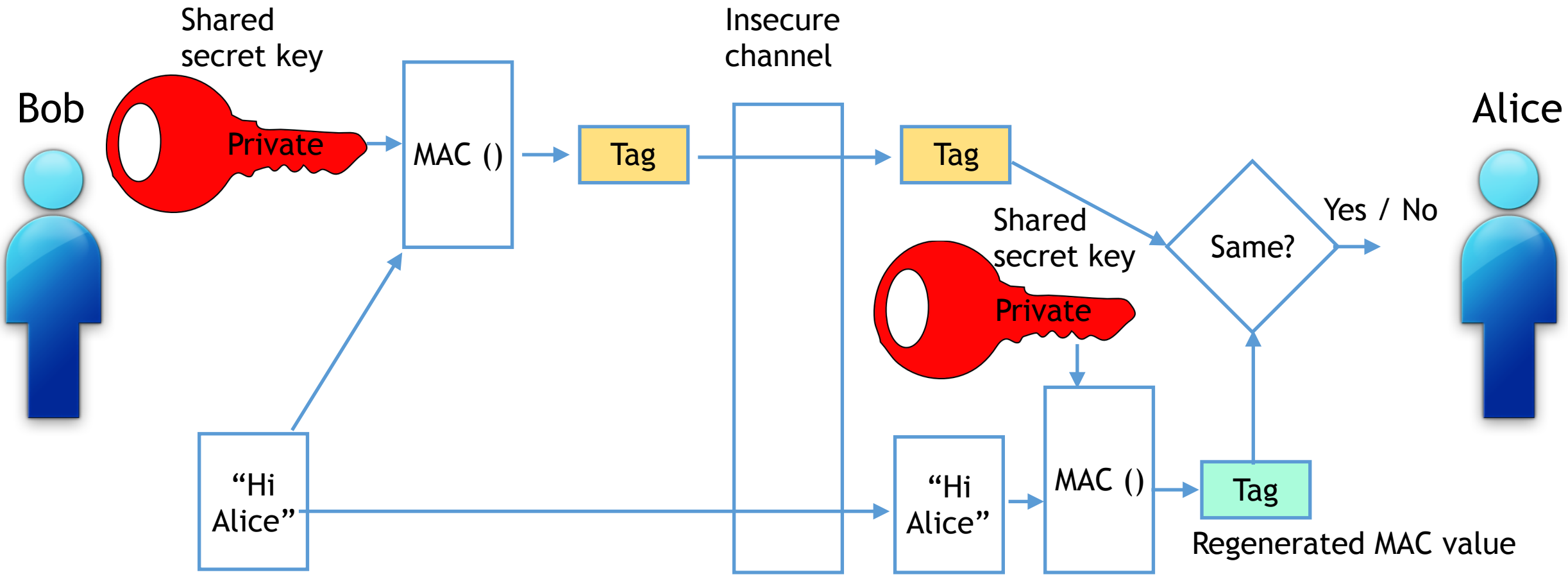
- Are there weaknesses?
- What about dictionary attacks?

# Message Authentication Codes (MACs)

- Accepts as inputs:
  - A secret key
  - A message of arbitrary length
- Outputs a fixed-size MAC value (also know as a **tag**)
- A cryptographic hash function is one method to generate a MAC value.
- See HMAC-SHA256, RFC 2104
- Often used for authentication

# MAC Usage

Here we are concerned about **authenticity** - did this message really com from Bob?
And **integrity** - has the message been modified in transit?



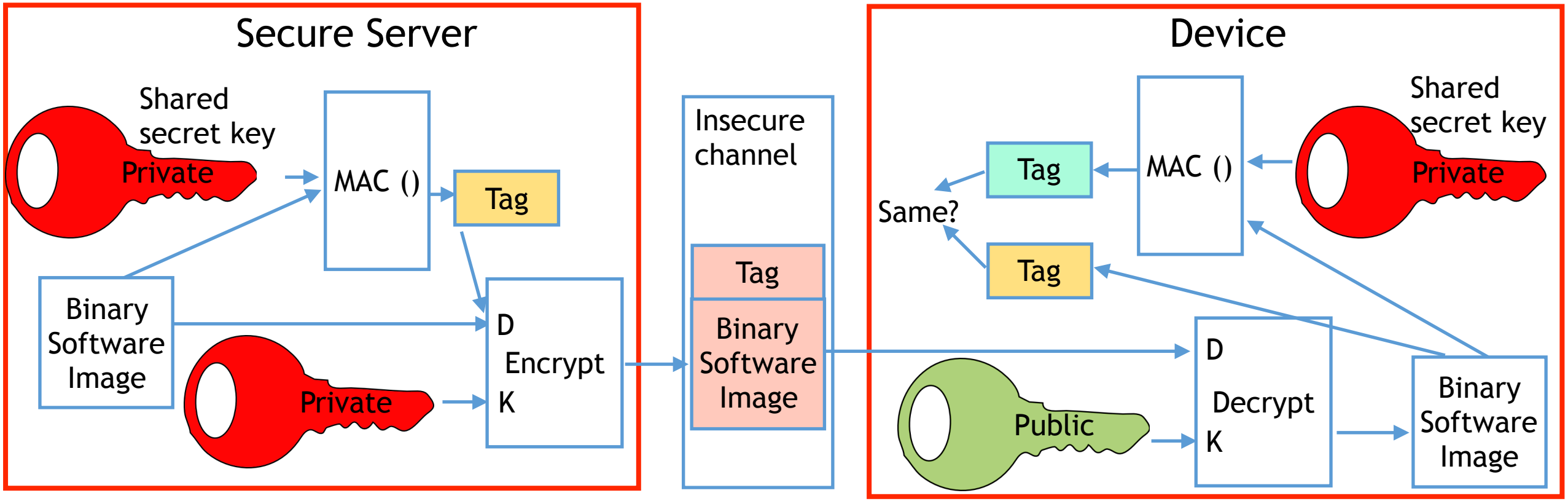
# Open source software downloads

- How many of you have seen/used an MD5 checksum?
- Vulnerabilities have been discovered that make it unsuitable for use as a cryptographic hash function, but can still be used for data integrity.
  - For example: Has anyone messed with the .zip file I downloaded?

# Software/Firmware Updates

- Should always be authenticated!
- Build this into the system from day one
- Asymmetric encryption would be a fair design choice

- Authentication
- Integrity
- Security



# Key Generation

- In order to be effective, the keys used for security should be random numbers so an adversary cannot guess them.
- My advice is to steer away from pseudo-random number generation because these have a predictable pattern.
- Very hard to generate “true” random numbers.
- Use the output or “measures” of physical systems that are inherently entropic, such that they are indistinguishable from “true” random numbers. Choices may include:
  - Temperature
  - Vibration
  - Flow rate
  - Ring oscillator
- Then mix these sources into your key generation

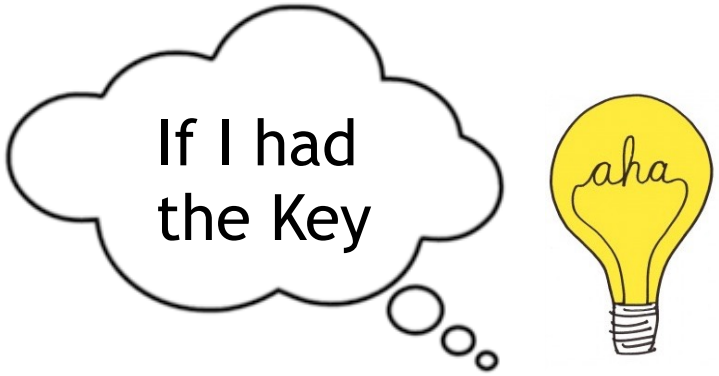
# Threat Vectors

- A threat vector (aka an attack surface) is an avenue that an attacker can exploit to gain access to a system, or to cause a system to leak information
- During the process of engineering systems, you have to ask the question: “What are we trying to protect against?”
- Be constantly applying the security mindset.

# Brute Force Attack on AES-256

- AES 256 has 256-bit keys
- 2^256 = $1.157 \times 10^{77}$ possibilities
- Using 1 trillion computers, and trying 1 key per picosecond, it would take
- $1.157 \times 10^{77} / 1 \times 10^{12} * 1 \times 10^{-12} = 1.157 \times 10^{53}$ secs or $3.7 \times 10^{45}$ years!

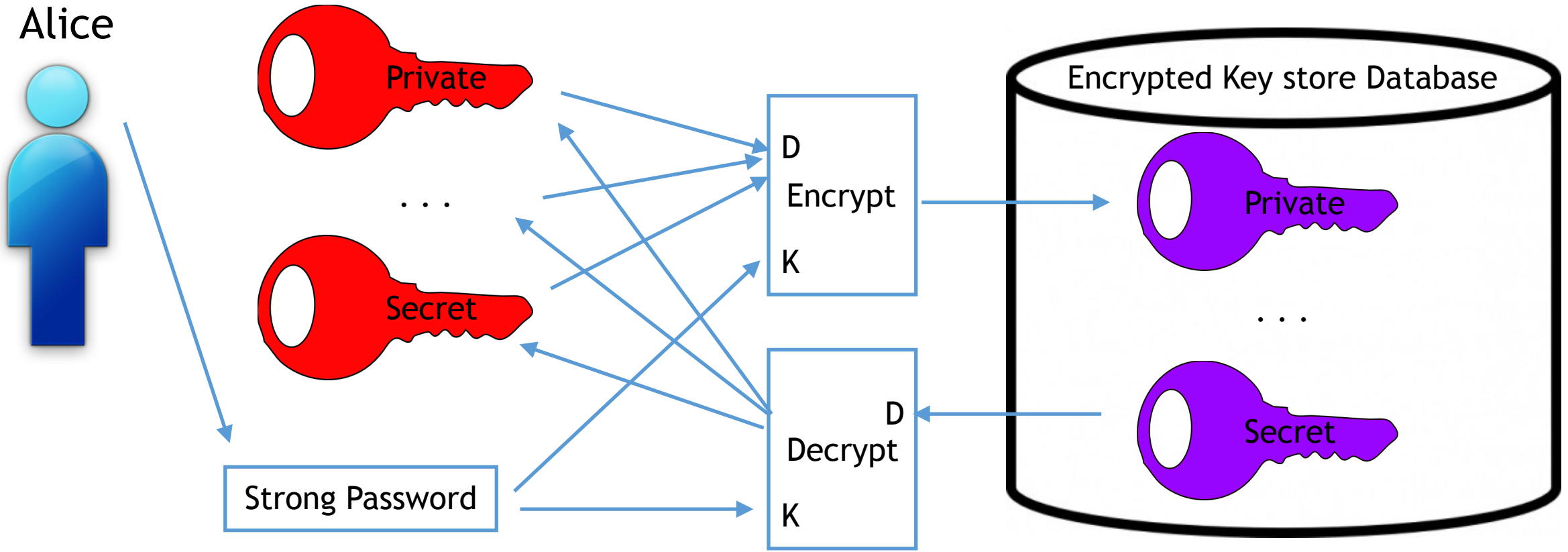
# Attacking AES




# Protecting The Key(s) - Key Management

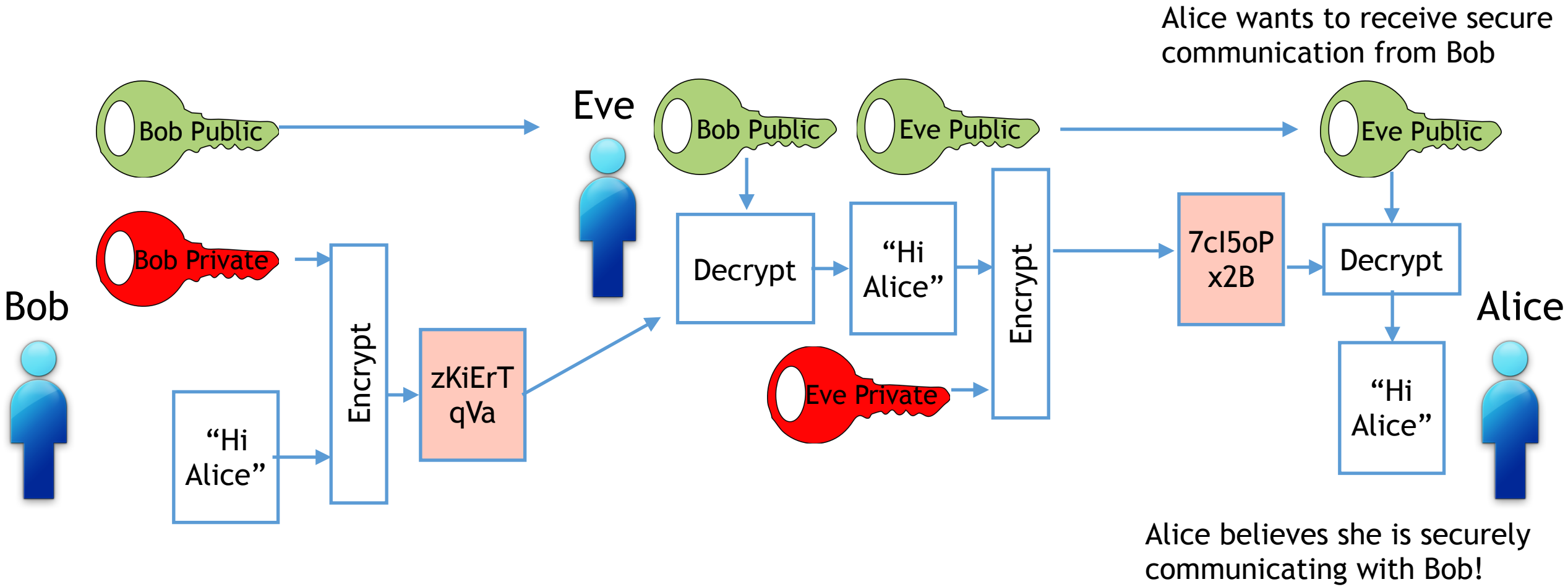
- Key management is the hard part
  - This is where you can get clever/creative
  - But never modify known algorithms
  - In order to get certifications (such as NIST or others) your device has to provide responses to known-answer tests : so trust the known algorithms
- Each key should have 1 purpose
- Vulnerability of keys increases:
  - The more you use it
  - The more places you store it
  - The longer you have it

# Protecting The Key(s)

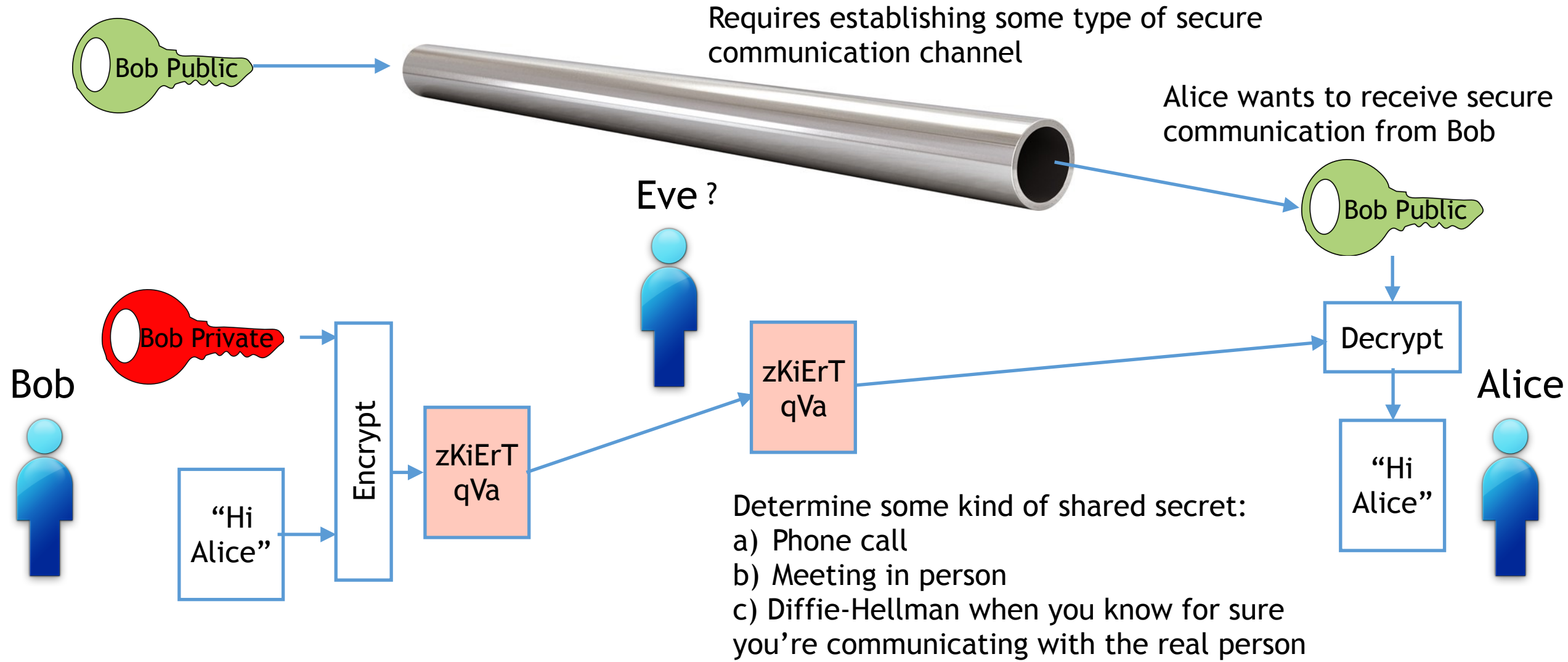



Private keys are “wrapped” with a password

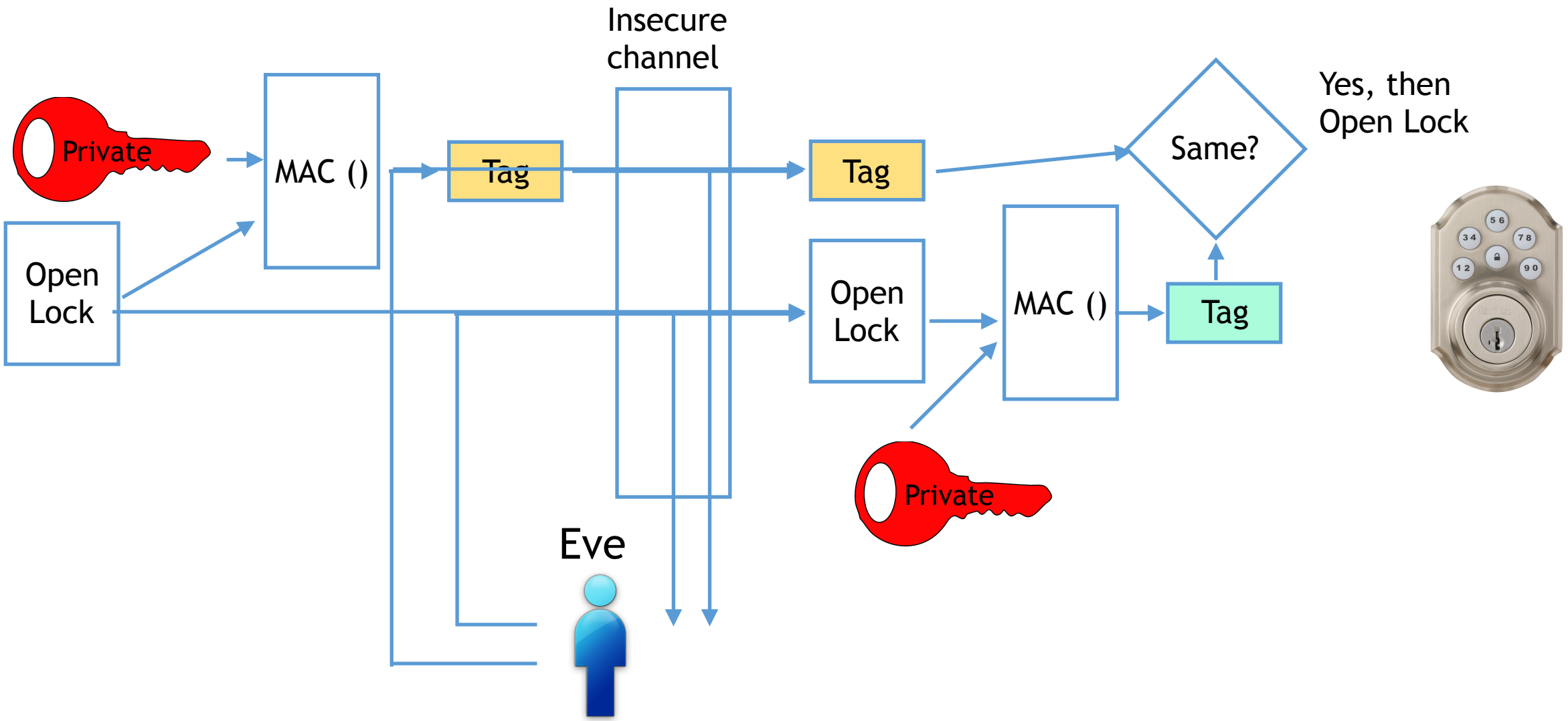
# Man in the middle attack



# Defeating Man in the middle attacks



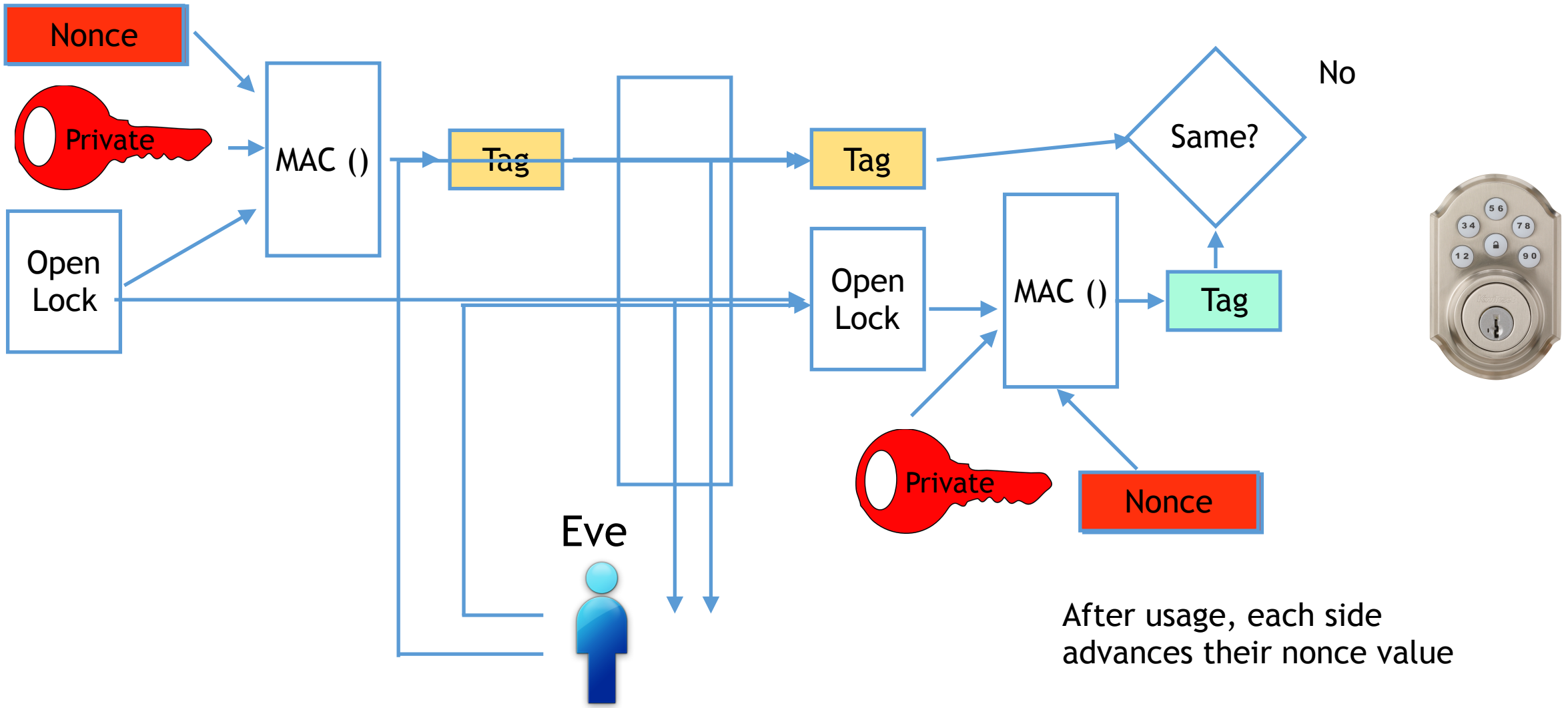
# Replay Attacks



# Nonce

- A nonce is a “Number used Once”.
- In its simplest form can be a counter value
  - But counter values might be guessed
- A better approach would be to use a pseudo-random number sequence. For example the output of a linear feedback shift register (LFSR). This might be guessed/ calculated as well.
- An even better approach would be to use an “unpredictable” sequence.

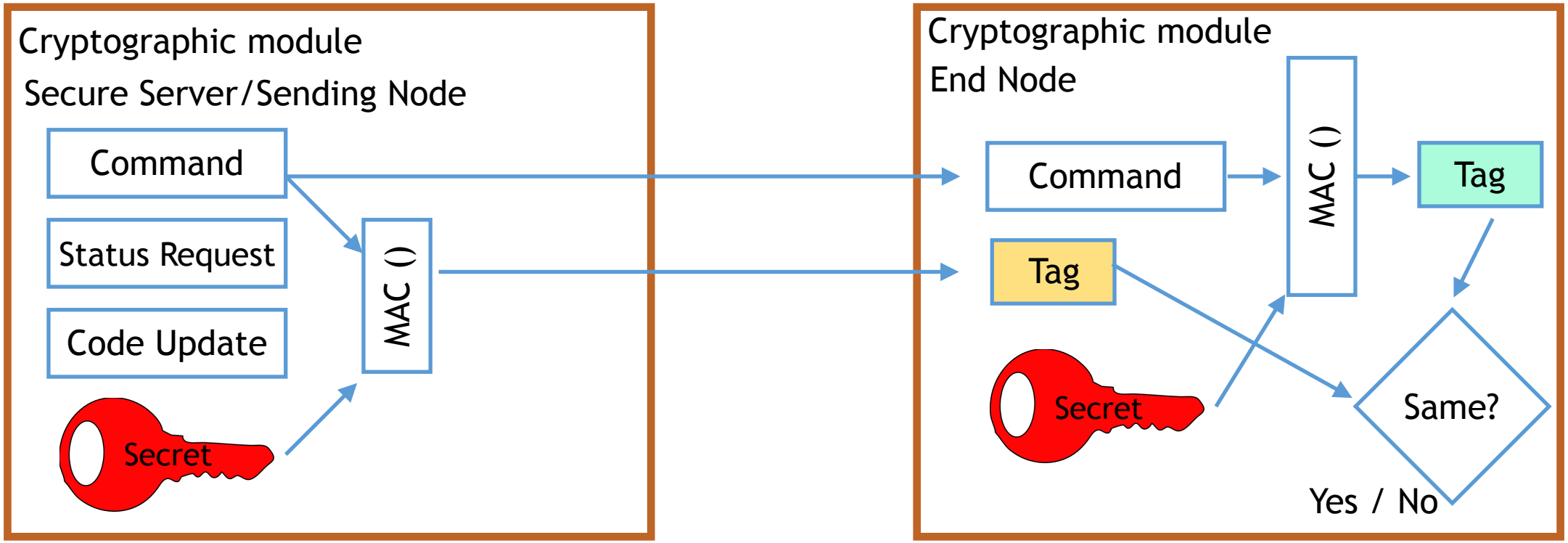
# Defeating Replay Attacks




After usage, each side advances their nonce value

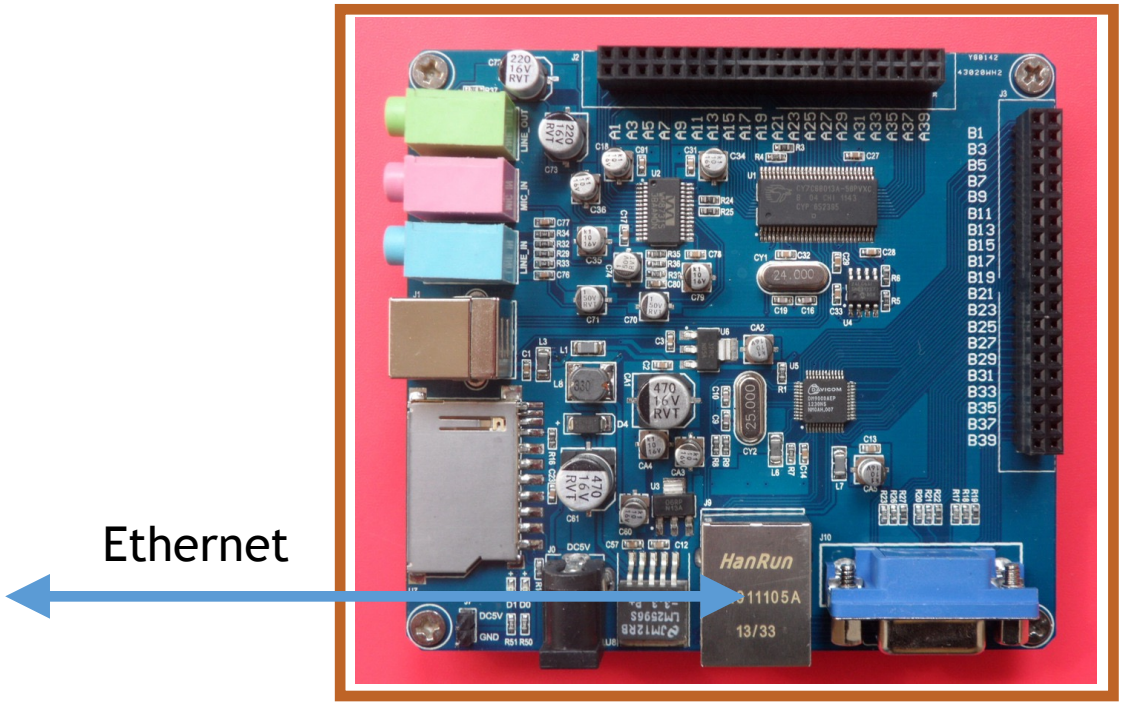
# Hardware Techniques

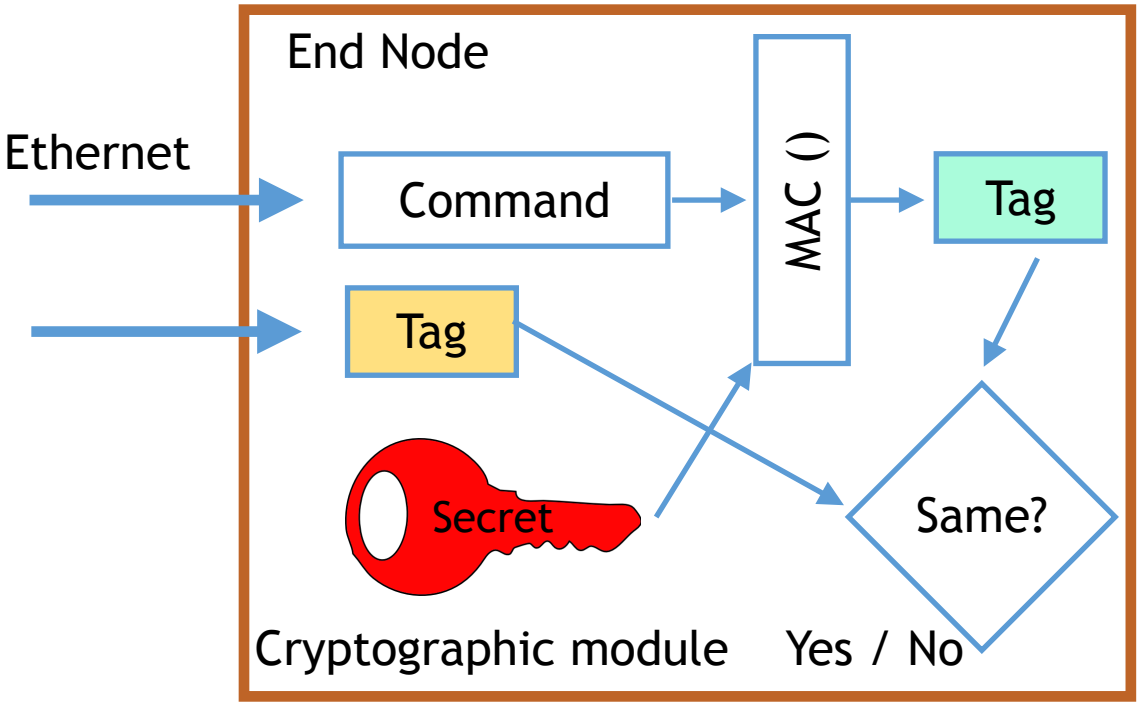
- Every communication channel in and out of a cryptographic module needs to have security concerns addressed.



# Hardware Techniques (con't)

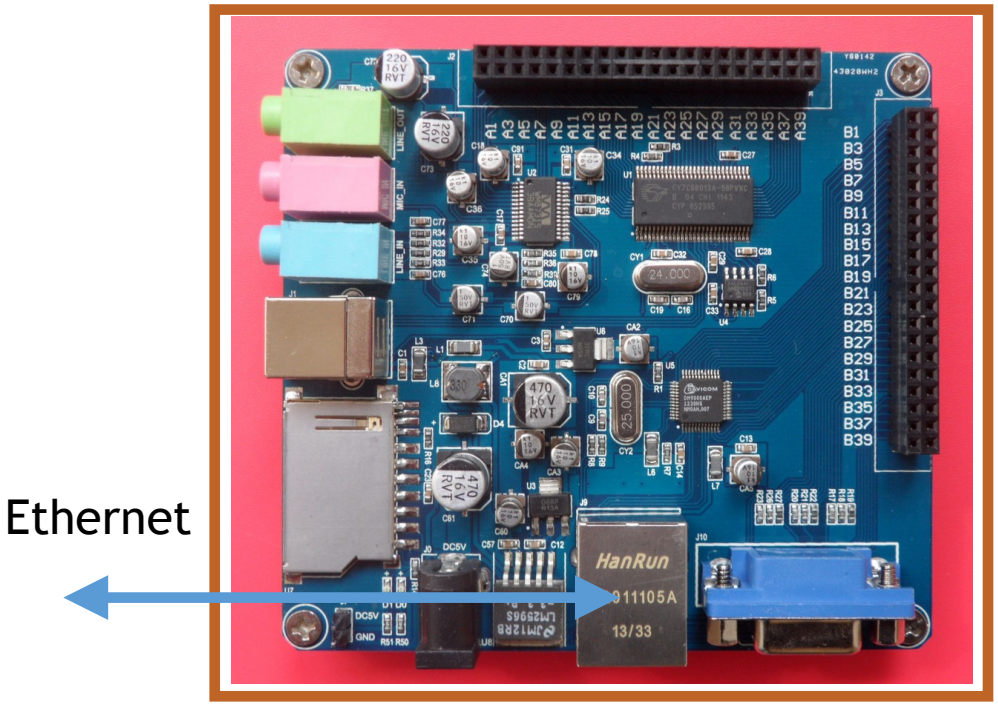
- Every communication channel in and out needs to have security concerns addressed.

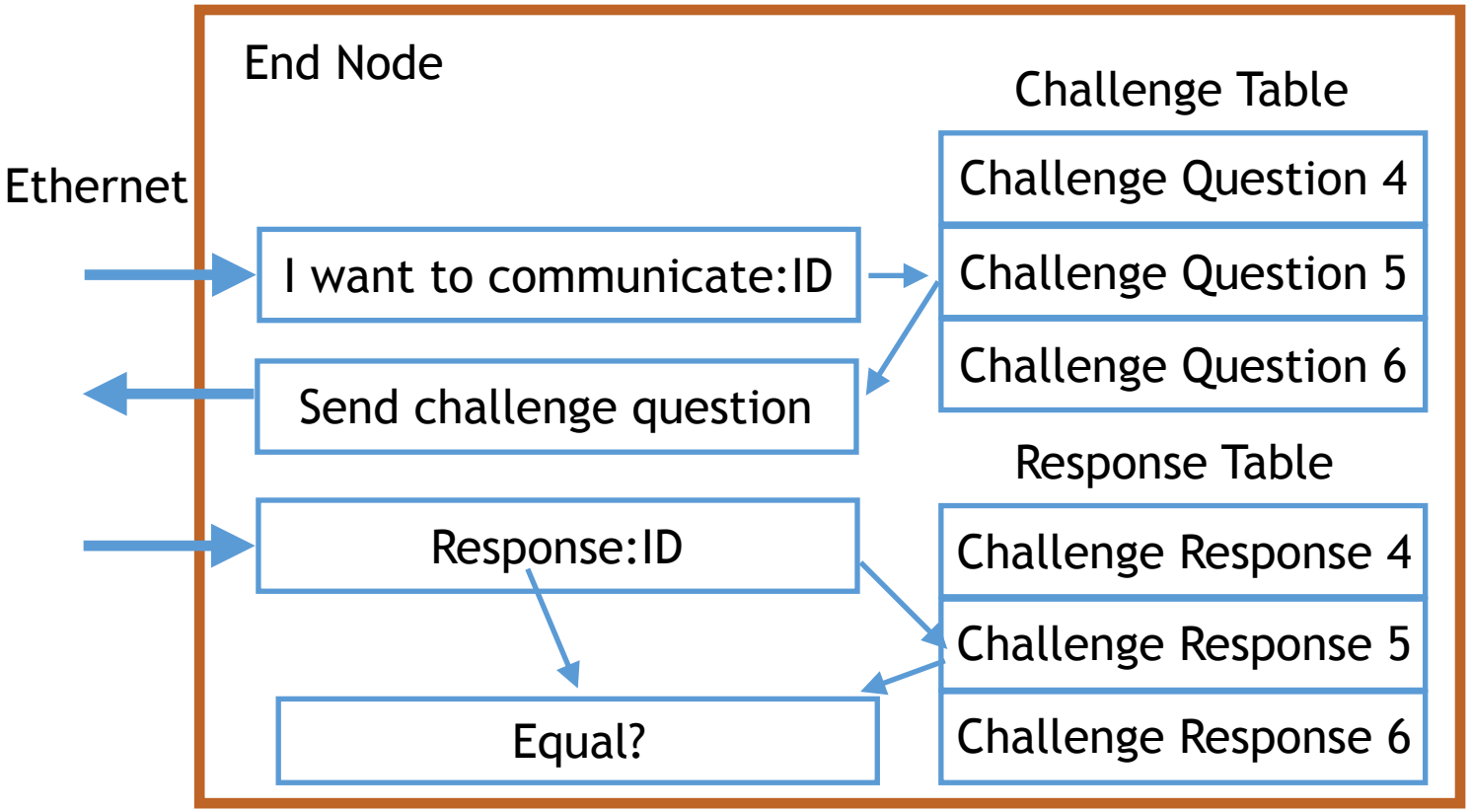


# Hardware Techniques (con't)

- Every communication channel in and out needs to have security concerns addressed.



# Additional Hardware Techniques

- Implement a Memory Protection Unit (MPU)
  - Run “application” code in user mode and the “kernel” in supervisor mode
  - Only supervisor mode has access to the MPU
  - Prohibits “rogue” application code from accessing memory of other processes
- Recently in the news Intel’s Spectre and Meltdown exploits, see this link

# Other Protection Methods

- Tamper evidence
  - Sticker
  - Epoxy resin (conformal coating), deterrent, but difficult for returned unit failure analysis
- Proximity detection, enclosure breach

# Software Techniques

- Every communication channel in and out needs to have security concerns addressed.
- Bounds check everything
- What happens when an attacker enters 1 million characters into a web login page - or a command packet arrives at your device that is a million times larger than what you specified as the maximum size in your hardware and software specifications?
- What about values on the stack?
- Apply security analysis to all levels of all protocols
- Conduct numerous design reviews with cross-functional team members

# Other Threat Vectors?

- Can you think of any unanticipated access channels an attacker might use?
  - Power
  - RF
  - Scan-chains
  - JTAG port
  - Temperature
  - ?

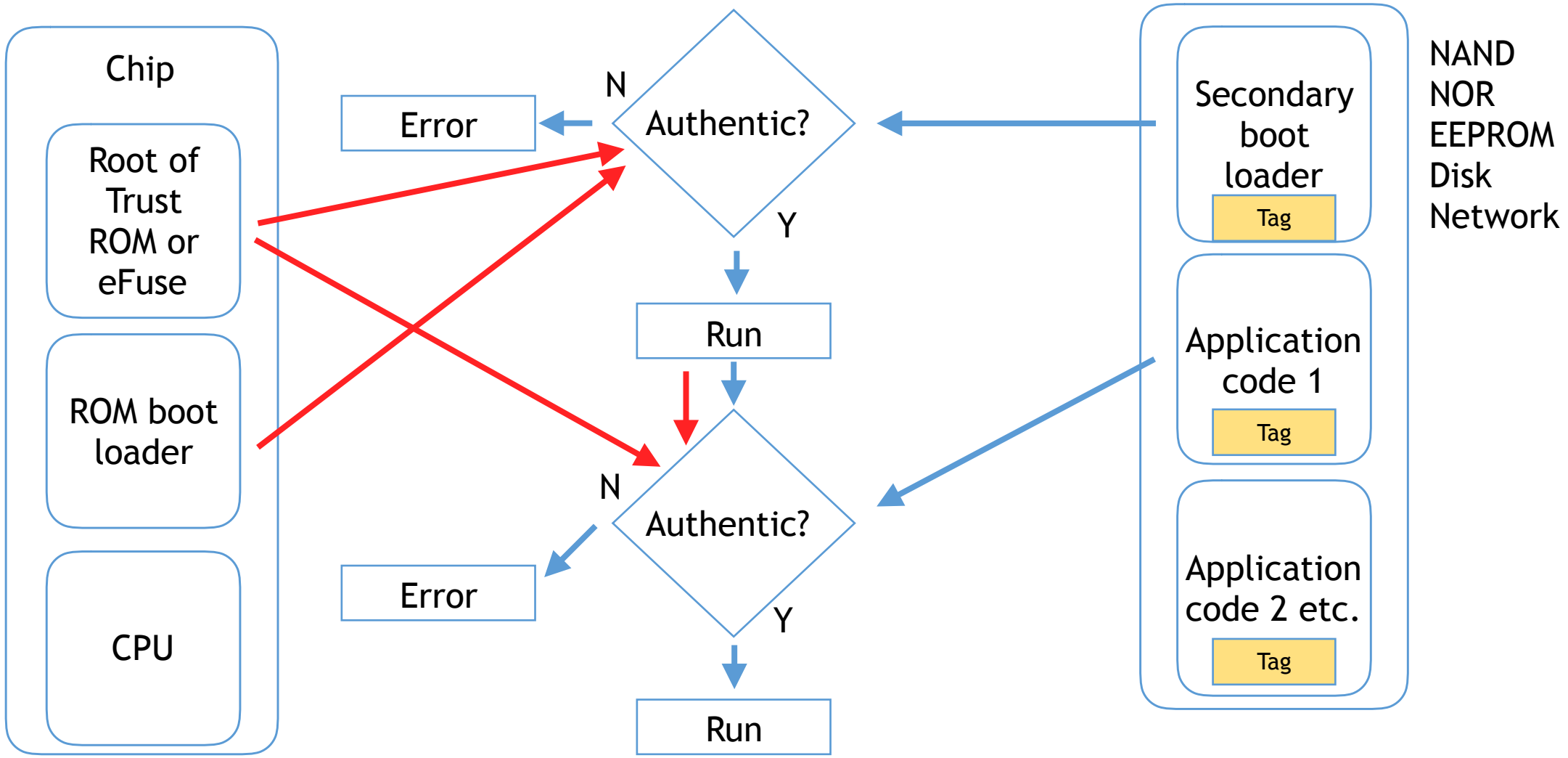
# Side Channel Attacks

- Differential power analysis
- RF analysis
- Power glitching
  - Puts hardware into an unknown and potentially compromised state
- Radiation
- See: Cryptography Research (a division of Rambus) for more information

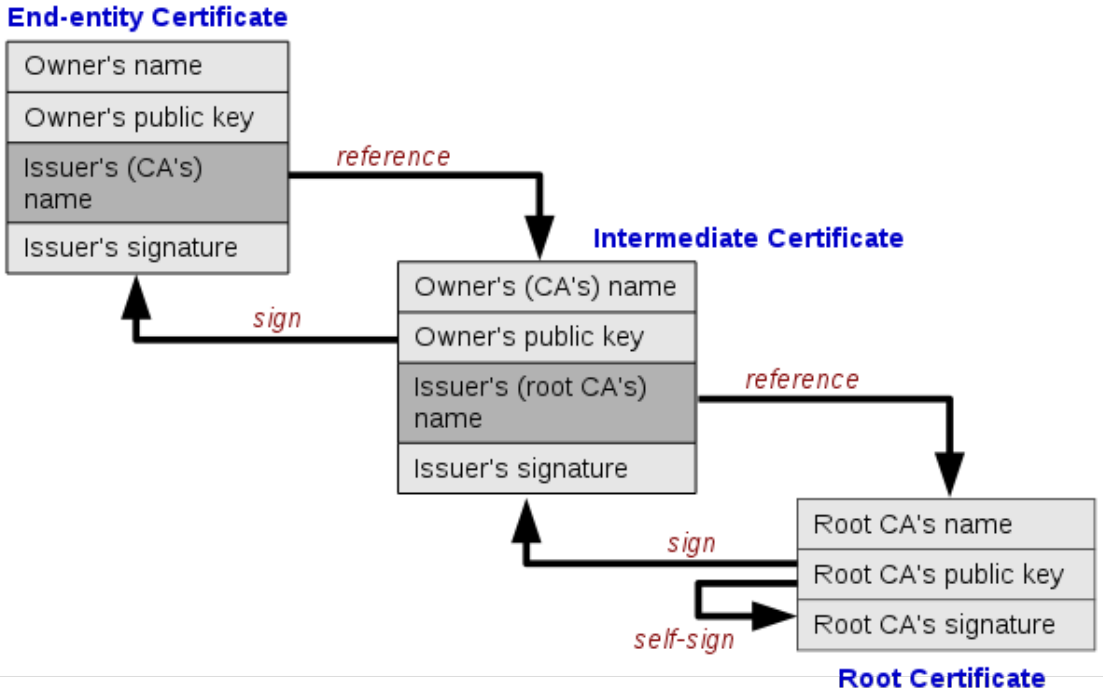
# Additional Info

- Cloud Security Alliance
- https://cloudsecurityalliance.org/download/security-as-a-service-working-group-charter/
- Bloomberg's 'What is Code?' article
  - https://www.bloomberg.com/company/announcements/bloomberg-businessweek-releases-code-issue-special-multi-platform-package-demystifying-code/
- Wall Street Journal
  - http://partners.wsj.com/bitdefender/history-of-hacking/watch-evolution-cyber-crime/

# Chain of Trust



# Chain of Trust (con't)




Source: https://en.wikipedia.org/wiki/Chain_of_trust

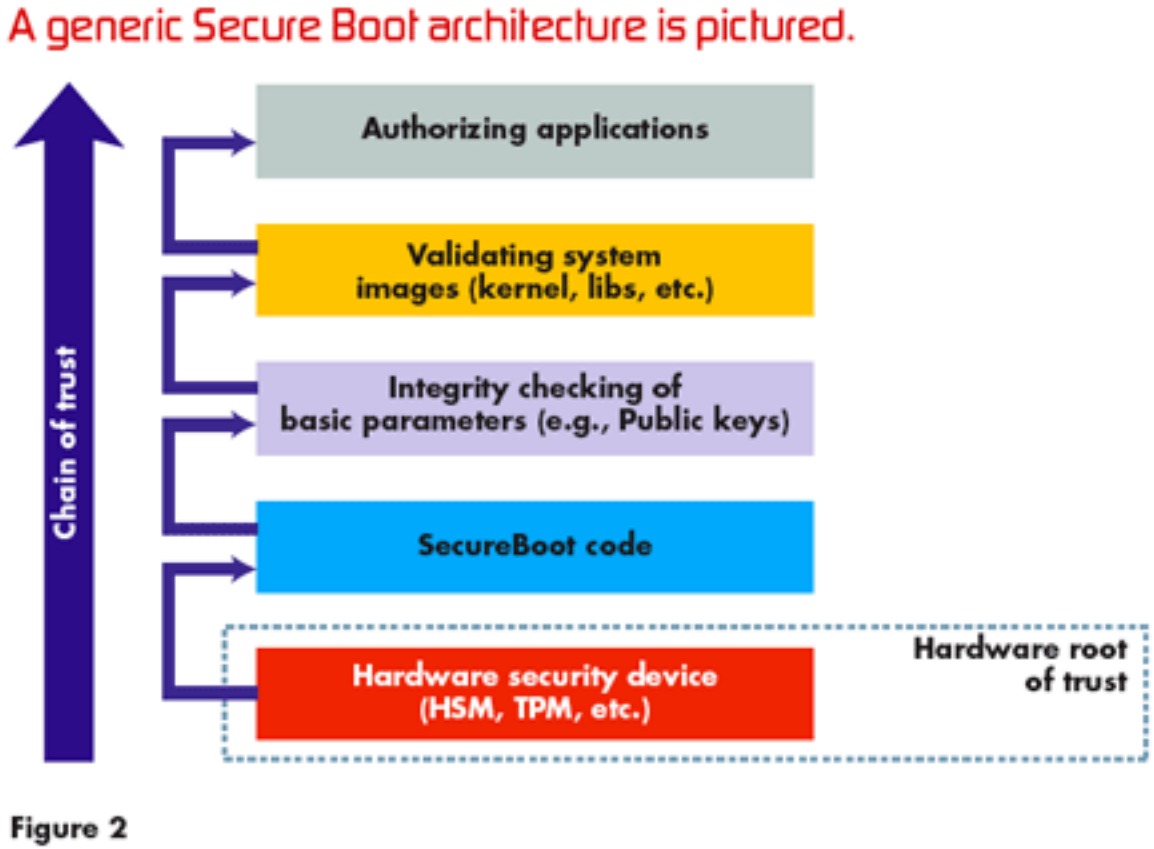



Figure 2

Source: https://www.embedded.com/design/prototyping-and-development/4007195/Employ-a-secure-flavor-of-Linux

# Examples of Poor Security Implementations

The following examples of compromised security come from "*Abusing the Internet of Things, Blackouts, Freakouts and Stakeouts*", first addition, by Nitesh Dhanjani

# Onity HT Door Lock


3 tracks
Onity uses track 3

Source: Cody Brocious http://demoseen.com/bhpaper.html

# Onity HT Door Lock - Data on a card key

**16-bit ident value**
Value assigned identifies which lock the card is associated with.
Employee master key = employeeID.

**8-bit flags byte**
Used for misc options.

**16-bit expiration date**
Defines length of time card is valid.

**24-bit unknown field**
Set to all 0's.

**24-bit keycode value + look-ahead value**
Locks are programmed with these values, ex: 100 and 50, lock would accept cards with values 100 to 150.
Every time a valid card is inserted, the lock resets its keycode value to the keycode value from the card, thereby invalidating older cards.
Keycodes representing master keys are also programmed into the locks.

# Onity HT Door Lock - Open Process

The fields on the mag strip are encrypted with a **sitecode**, a random 32-bit value, assigned by the manufacturer to identify a specific property. Encryption algorithm is custom.

Process

Card swiped → Data is decrypted using the sitecode → ident and expiration date is checked → keycode value is checked and if in look-ahead range the lock opens. Lock saves keycode value

What data does the lock have?

- sitecode
- ident value (which lock am I?)
- keycodes for master keys
- date and time
- most recent keycode value from last card swipe/lock open operation

# Onity HT Door Lock - Portable Programmer

```
00000000 73 69 74 65 63 6f 64 65  | s  i  t  e  c  o  d  e
00000008 20 30 46 31 33 42 43 33  | sp 0  F  1  3  B  C  3
00000010 30 0a 69 64 65 6e 74 20  | 0  nl i  d  e  n  t  sp
00000018 33 46 34 35 0a 6d 61 73  | 3  F  4  5  nl m  a  s
00000020 74 65 72 6b 65 79 20 46  | t  e  r  k  e  y  sp F
00000028 37 38 32 31 39 0a 6c 61  | 7  8  2  1  9  nl l  a
00000030 73 74 20 6b 65 79 63 6f  | s  t  sp k  e  y  c  o
00000038 64 65 20 32 33 34 31 38  | d  e  sp 2  3  4  1  8
00000040 30 0a 20 0a              | 0  nl sp nl
```

$50 Arduino

Portable Programmer

dump memory

open cmd

Programming port

sitecode

master keycode

make master keys

# Consequences

- Many burglaries were reported
- Tarnished the reputation of:
  - Onity
  - Hotels

# Onity's Response

- Issued a response that they would:
  - Place a mechanical cap over programming port, torx screw
  - Update the firmware
- Issues:
  - Torx wrenches are easy to obtain
  - Design of the PCB needed to be updated to enable secure firmware update
- Onity apparently quietly worked with hotel chains to replace the PCB's
  - Millions of PCBs replaced, at what cost?

# Z-Wave Door Lock


Z-Wave controller

Source: Behrang Fouladi, Sahand Ghanoun

# Z-Wave Door Lock


Researcher's controller

Z-Wave controller

1. Device is unpaired, initial key exchange is performed
2. Common key determined, stored in EEPROM, device now paired
3. *Frame encryption* key generated, used to encrypt payloads in subsequent communications
4. *Data origin authentication* key generated used to generate a MAC value to address replay attacks

The flaw:
The researchers could transmit a new key-exchange packet.
The **lock firmware** failed to check if there was already an existing common key in EEPROM,
and so went through the key exchange sequence again, enabling an attacker to pair and unlock the lock.

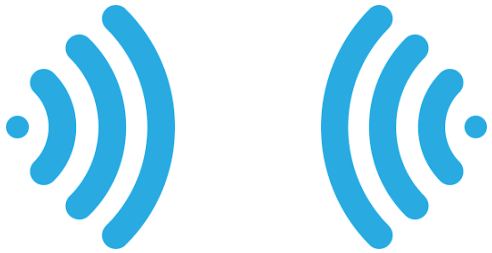
Source: Behrang Fouladi, Sahand Ghanoun

# Bluetooth BLE Vulnerabilities


Source: Mike Ryan whitepaper: “Bluetooth: With Low Energy Comes Low Security”
https://lacklustre.net/bluetooth/Ryan_Bluetooth_Low_Energy_USENIX_WOOT.pdf

# Bluetooth Vulnerabilities

Bluetooth BLE
Slave



Bluetooth BLE
Master

Master and slave can use encryption to secure data exchange.
Must establish a shared secret known as the Long-Term-Key (LTK).

The exchange process starts by selecting a temporary key (TK).
According to the BLE specification, TK=0 if “Just Works” mode is selected.
Just Works mode is used for devices with no display/input capability. Otherwise a TK value from 0 to 999999 is used.
Once the TK is established, the devices establish a Short-Term-Key (STK), and eventually establish a LTK.

# Bluetooth Vulnerabilities

Ryan created a tool called Crackle.
Ryan captured the BLE data exchange and input the data to crackle.
Crackle attempts to brute-force the TK by choosing values from 0 to 999999.
Once the TK is found, the STK can be found by decrypting it with the TK. Then the LTK can be found by decrypting it with the STK.

The flaw:
The range of TK's is relatively small. **In this case it was practical to try every key**.

Devices don't have to use the "Just Works" BLE specification and can rely on schemes where the keys are 128-bits.

# Incidents In The News



Sony's Settlement With Employees Over Hacked Data Worth More Than $5.5 Million

Critical Unpatched Flaws Disclosed In Western Digital 'My Cloud' Storage Devices

Friday, January 05, 2018 Swati Khandelwal


tom's HARDWARE

SECURITY > NEWS

Intel AMT Allows BitLocker Bypass In Under A Minute

41 percent of Android phones are vulnerable to 'devastating' Wi-Fi attack

*Every Wi-Fi device affected by some variant of attack*

Windows Central

PATCH IT UP

Lenovo discloses security vulnerability in ThinkPad fingerprint manager

Lenovo's fingerprint management software for Windows 7 and 8 has a pretty major vulnerability, but a patch is available.


Seagate Quietly Patches Dangerous Bug in NAS Devices


The hackers who broke into Equifax exploited a flaw in open-source server software

CR Consumer Reports

Major Security Flaw Found in Netgear Routers

Samsung Left Millions Vulnerable to Hackers Because It Forgot to Renew a Domain, Researchers Say

The Register®

Biting the hand that feeds IT

DATA CENTER SOFTWARE NETWORKS SECURITY INFRASTRUCTURE BUSINESS HARDWARE SCIENCE BOOT

Security

Western Digital's hard drive encryption is useless. Totally useless

46

Rookie errors make it child's play to decrypt data

Security

Qualcomm joins Intel, Apple, Arm, AMD in confirming its CPUs suffer hack bugs, too


Target To Pay $10 Million To Settle Lawsuit From Massive Data Breach

# Incident Response

- If your company's products or services are hacked/compromised, what do you say:
  - To your Employees?
  - To your Customers?
  - To the Public?
- Important to prepare a response(s) in advance
- NIST: https://www.nist.gov/el/intelligent-systems-division-73500/incident-response-scenarios
- Microsoft: https://www.microsoft.com/en-us/cybersecurity/default.aspx

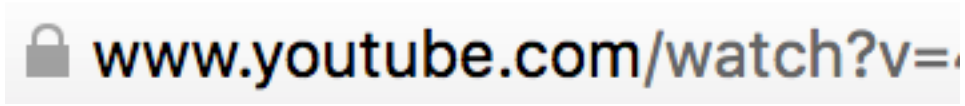
# Web Browsers

# TLS / SSL

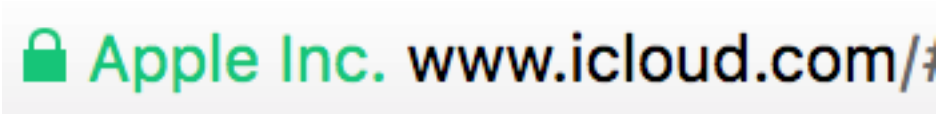
- Transport Layer Security
- Secure Socket Layer (predecessor to TLS)
- When a connection is secured by TLS, in our case a web browser and a server, the connection will have one or more of the following properties:
  - The connection is ***private*** (secure) via symmetric encryption
  - The identity of the communicating parties can be ***authenticated*** using public-key encryption, they are who they say they are
  - The connection has ***integrity*** because each message exchanged is protected by a MAC to detect alteration

# Web Browser Examples

Encrypted and authenticated (standard certificate)




Encrypted and authenticated (extended validation (EV) certificate)




Encrypted, not authenticated


Not encrypted

# What is a Digital Signature and Certificate

Company that owns the web server creates a certificate and sends to a CA to be “signed”


Certificate Authority (CA)

CA Private

Cryptographic Hash

digest

Encrypt

signature

insert signature


Back to Requesting company

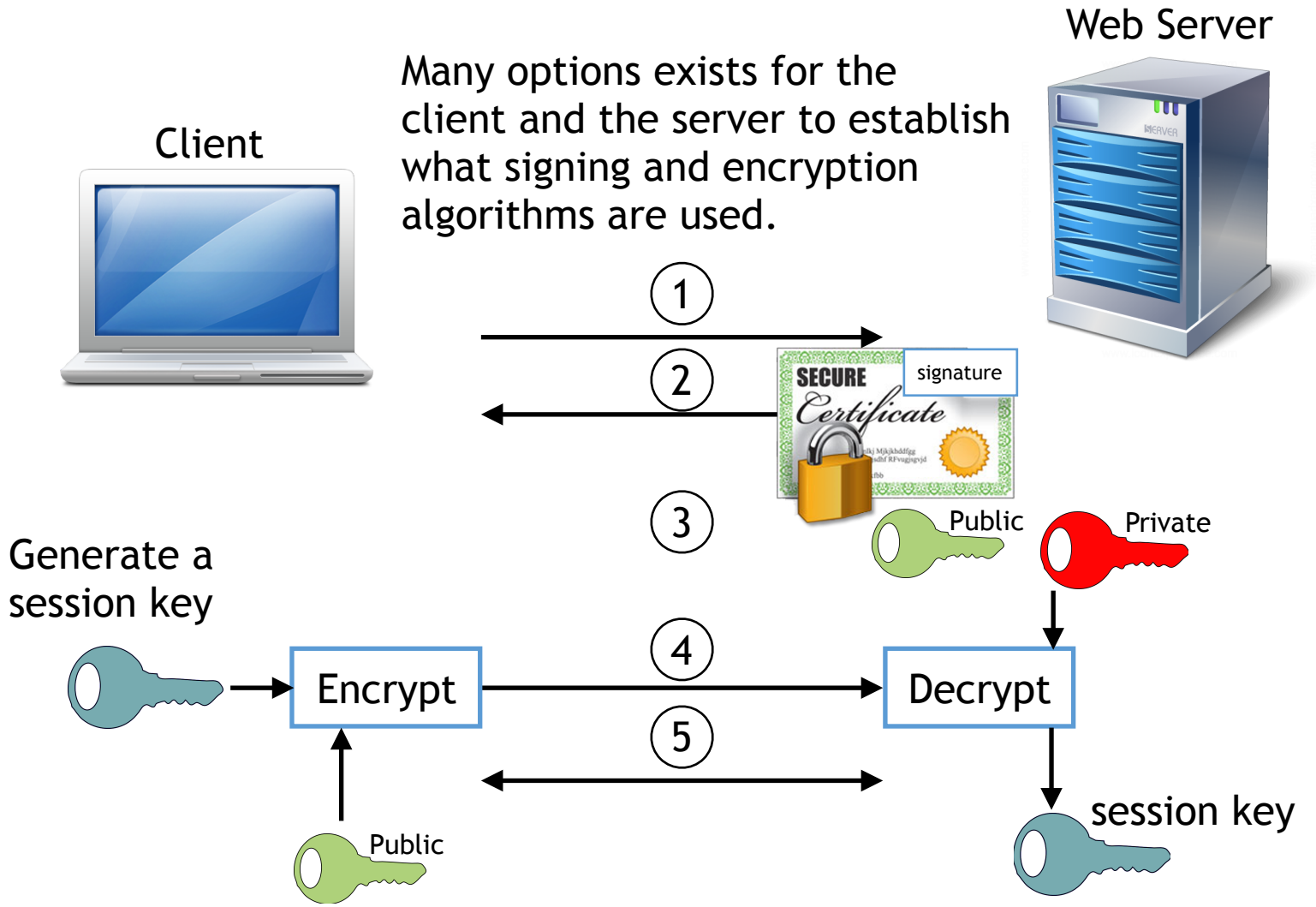
Make key publicly available

CA Public

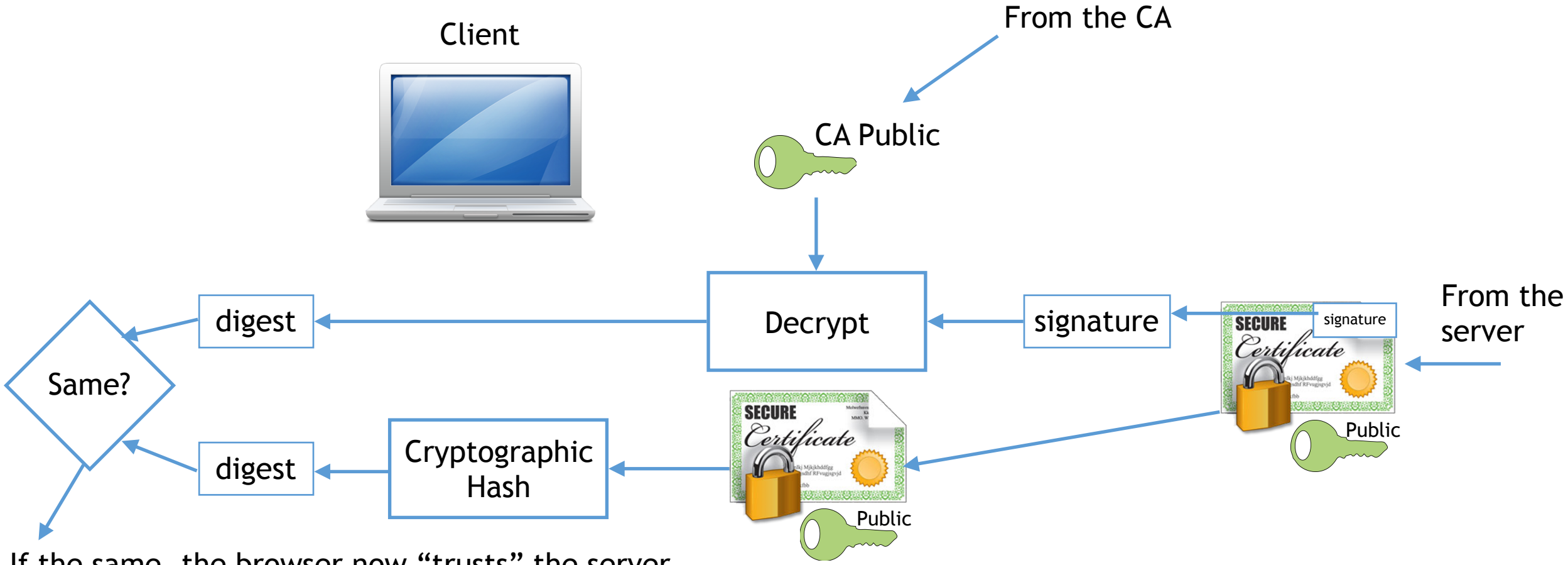
Source: http://searchsecurity.techtarget.com/definition/digital-signature

# Steps

- 1 Client makes an SSL connection request
- 2 Server responds with an SSL certificate
- 3 Client validates (Authenticates) the certificate (next slide)
- 4 Client generates a symmetric session key, sends to server
- 5 SSL session is established using the session key, used for subsequent communication



# Step 3 - Authenticate Certificate



# Certificate Examples

**com.apple.idms.appleid.prd.4c6b39327675464958464e366c627530624978635 4773d3d**
Issued by: Apple Application Integration Certification Authority
Expires: Thursday, June 22, 2017 at 7:07:06 PM Mountain Daylight Time
This certificate is valid

▶ Trust
▼ Details

| | |
|---|---|
| **Subject Name** | |
| Common Name | com.apple.idms.appleid.prd.4c6b39327675464958464e366c627530624978635 4773d3d |
| **Issuer Name** | |
| Country | US |
| Organization | Apple Inc. |
| Organizational Unit | Apple Certification Authority |
| Common Name | Apple Application Integration Certification Authority |
| Serial Number | 7262869206556401725 |
| Version | 3 |
| Signature Algorithm | SHA-256 with RSA Encryption ( 1.2.840.113549.1.1.11 ) |
| Parameters | none |
| Not Valid Before | Tuesday, June 23, 2015 at 7:07:06 PM Mountain Daylight Time |
| Not Valid After | Thursday, June 22, 2017 at 7:07:06 PM Mountain Daylight Time |
| **Public Key Info** | |
| Algorithm | RSA Encryption ( 1.2.840.113549.1.1.1 ) |
| Parameters | none |
| Public Key | 256 bytes : F5 AC 80 24 45 65 A2 86 8A 57 2C B2 D1 1A 61 97 92 42 6E 24 6B 0B 1E FF B3 C5 B8 3A 61 A8 38 E0 32 37 CB 60 13 71 25 EA F0 CC D7 F5 36 1C 5F 44 6E C6 52 69 AC 75 49 C5 96 1D F5 98 99 FD 3F 9E CD 52 94 29 C3 6F D2 94 6B 37 22 DB BD 69 83 D4 D3 69 E6 07 46 65 23 BA DC AB C8 32 FA D1 CF 95 79 5D 5F 79 3E 23 8B 85 63 3D 36 46 E0 5C AE A2 0E F8 95 6B 90 74 26 1D 77 1B C7 6C 04 27 74 2A 8F 0D 7F 5E F6 2C A0 0F 10 A1 6F 44 E9 F5 33 51 34 B9 15 1D 6B 8E CA 07 67 4F 12 74 48 38 75 7F 9C 84 EB B4 84 7E E1 3B 17 13 63 DE 13 B8 6F 39 BD 2E ED 45 58 36 54 E7 51 41 A5 CB D4 A7 D9 D3 B3 00 76 5F 46 C5 B1 03 D9 BD AE BF 44 93 AB 70 A3 B2 68 7F 5D 27 56 7B B1 D0 03 F8 F6 2D 91 BA F9 95 09 D7 36 4F 33 BB DD 91 15 A9 F8 C7 38 E6 61 38 DC DF A9 6F F2 A7 94 25 71 28 40 62 5F FF |
| Exponent | 65537 |
| Key Size | 2048 bits |
| Key Usage | Encrypt, Verify, Derive |
| Signature | 256 bytes : A6 F1 44 0A A0 4C 96 82 08 59 3C 2F FB 24 35 37 68 AE 27 D3 3F 78 F5 2F B1 0E 3C 05 14 50 37 C4 5E 4B 53 8E 30 42 88 36 00 CA 82 2A 37 D1 C3 C6 B5 05 6C 6F 8E B8 F0 02 F5 01 74 40 21 AE 1C 1B 44 C4 D4 39 95 1F C5 73 30 F2 E5 4B 96 72 40 4B 7F 94 EE 71 8D D5 66 63 A9 4A 24 12 6F E0 6F B8 F2 DE 43 D1 3F 7B 1A 7F BC 14 15 37 53 DF C0 98 33 2E FC 50 AE 27 E0 19 12 02 F1 7E E4 6C DC 09 54 A2 0E 22 01 FF 2D 9D 2B A2 BE 3A BD 25 92 61 3D 70 FF 62 F3 F0 9B 29 97 DF 65 B5 FB 18 C5 5D C8 9B 01 A2 0A 7E A9 44 90 39 3E 1B 55 09 19 A5 A2 65 17 BF 68 07 80 2A F4 F5 2D 60 2D F8 14 CE 9B FC AD E9 B3 A6 BD A1 29 72 59 EA 0D C2 B7 77 48 57 34 27 D9 B5 6F 2B 77 75 A9 57 BE D4 B0 8B 20 B7 37 A3 C6 F8 D8 3F 02 E9 05 EC 89 B9 9B 68 A0 2D BD F4 00 1A 89 C6 2A EA DE E3 93 E2 B4 BF |

**Lauterbach MAIL CA**
Intermediate certificate authority
Expires: Sunday, October 11, 2020 at 7:37:47 AM Mountain Daylight Time
This certificate was signed by an unknown authority

▶ Trust
▼ Details

| | |
|---|---|
| **Subject Name** | |
| Country | DE |
| State/Province | Bavaria |
| Organization | Lauterbach GmbH |
| Common Name | Lauterbach MAIL CA |
| **Issuer Name** | |
| Country | DE |
| State/Province | Bavaria |
| Locality | Hoehenkirchen |
| Organization | Lauterbach GmbH |
| Common Name | Lauterbach ROOT CA |
| Serial Number | 2 |
| Version | 3 |
| Signature Algorithm | SHA-1 with RSA Encryption ( 1.2.840.113549.1.1.5 ) |
| Parameters | none |
| Not Valid Before | Thursday, October 14, 2010 at 7:37:47 AM Mountain Daylight Time |
| Not Valid After | Sunday, October 11, 2020 at 7:37:47 AM Mountain Daylight Time |
| **Public Key Info** | |
| Algorithm | RSA Encryption ( 1.2.840.113549.1.1.1 ) |
| Parameters | none |
| Public Key | 128 bytes : D3 38 D9 FB AF 02 35 1A 7F 18 D1 B2 FF DB 31 5E 81 18 3E 85 09 22 46 18 F6 C5 80 28 50 16 DF B0 1C 8F 58 36 39 CD 2A A6 3B 63 48 4D ED 72 38 52 7B 2B A6 E8 89 87 AD ED 6C 72 1E 52 C2 E3 5C 55 5F 63 6E 25 71 DA C6 ED 1D F6 F4 B8 BC 8E B0 F7 7B FE 81 ED D0 15 0F E3 4D 73 AB ED 81 AB A1 A6 63 03 22 B5 82 0C 45 CB EF AB EA 10 DF A5 A2 0E 3C 84 11 8A 6D 6A 3F 86 61 32 16 B8 35 DF C3 E7 |
| Exponent | 65537 |
| Key Size | 1024 bits |
| Key Usage | Any |
| Signature | 128 bytes : 98 45 BF AA 77 D3 13 86 4D 85 62 07 8C 91 78 B8 D4 BB 96 86 6A 07 21 6B FF B4 DF 19 67 34 0F E2 C3 94 9B D2 F1 BC 41 1E 07 DC 51 D6 8D 65 E1 F0 37 60 3C 25 0F 06 05 74 1C BB 5F 90 C8 89 67 65 A4 6B DE 0C 39 52 3B 1B CC 15 FD 03 7D 05 28 41 6A 57 7B 92 24 6B CA 7E BE 4C 90 EB BC 0E 65 67 05 0E E0 5A D1 07 9F F0 D7 CA D2 4D C9 22 97 E6 77 A9 25 0A 68 1E 31 76 01 65 59 DC 1D C6 3E 2B |

# TLS/SSL Summary

- Certificates are issued by trusted organizations such as VeriSign or RSA Security, known as Certificate Authorities
- Example:
  - Your bank requests a certificate from VeriSign, includes the bank's public key
  - VeriSign confirms identity of the bank/server, creates a certificate and signs it. Gives it back to the bank for the bank's server to hand out
  - Browser receives certificate and checks the certificate's authenticity
  - Trust is established

# Blockchains

# Blockchains

- In 2009 a hacker (or group of hackers) known as Satoshi Nakamoto unveiled the world's first digital currency
- The technology works on the principle that at its foundation, money is just an accounting tool
- It defines a method for:
  - Abstracting value
  - Assigning ownership
  - Providing a means for transacting

Source: IEEE Spectrum, October 2017

# Blockchains (con't)

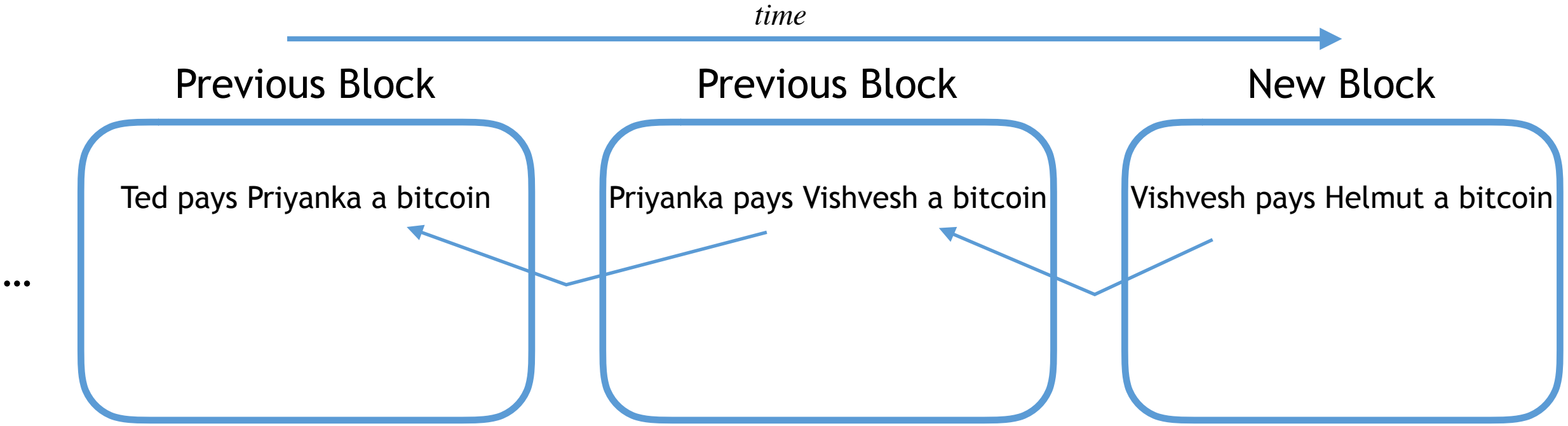
- Cash has been the historical means
- Processing the physical tokens (coins, bills) equals ownership
- It's up to individuals to negotiate transactions among themselves
- As long as cash is sufficiently difficult to replicate, there is no need for a **complete accounting** of who owns what portion of the money supply, or for the details of who the various holders were of a single $10 bill going all the way back to when it was printed

# Blockchains (con't)

- If you could piece together a running tabulation of who held every bill, then the physical representation would become unnecessary
- Banks and payment processors have partially sublimated physical currency into digital records within their closed systems
- **Bitcoin** completed the transformation by creating a **single, universally accessible digital ledger**, called a **blockchain**
- It's called a chain because changes can be made only by adding new information to the end

# Blockchains (con't)

- Each new addition (a block) contains a new set of transactions. These new transactions reference previous transactions in the chain



# Blockchains (con't)

- Bitcoin's block chain (i.e. ledger) is replicated on networked computers around the globe
- Accessible to anyone with a computer and an internet connection
- A class of participants on this network, called **miners**, are responsible for:
  - Detecting transactions
  - Validating the transactions
  - Adding them to the blockchain as new blocks

# Blockchains (con't)

- Validation entails:
  - Verifying that a person actually owns the bitcoins in a transaction
  - Verifying those bitcoins have not been spent elsewhere
- Ownership on the bitcoin blockchain is determined by a pair of public/private keys
  - The public key resides in the blockchain for anyone to see
  - The owner keeps the private key private

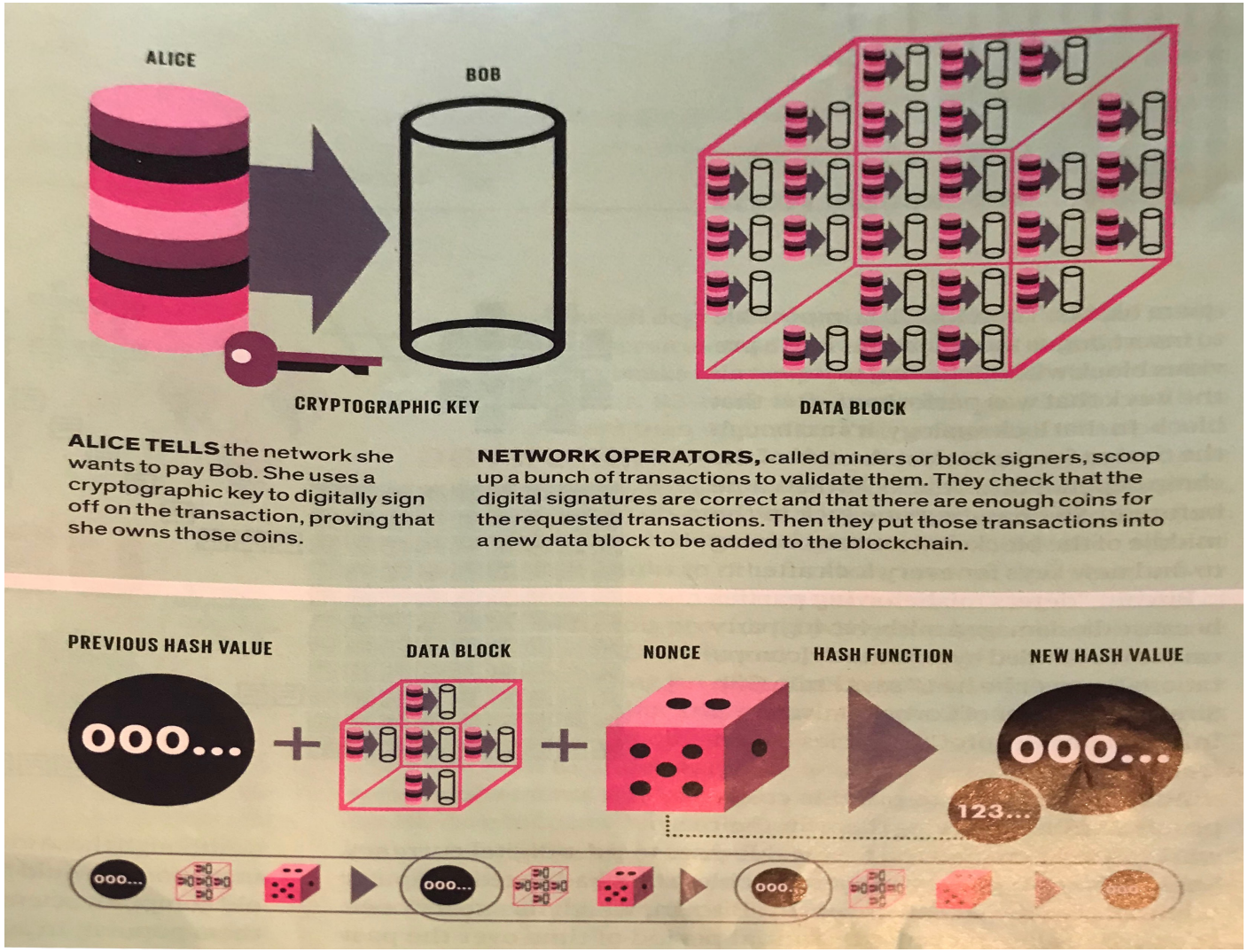
# Blockchains (con't)

- For Helmut's transaction:
  - The transaction is combined with the private key
  - Calculations are performed producing a long number
  - Anyone who has the original transaction and knows the public key can then do some calculations of their own to prove that the long number was in fact created with the private key

# Blockchains (con't)

- The main role of miners is to insure the **irreversibility** of new transactions, making them final and tamperproof
- "*The method they use for doing so is thought to be the most significant contribution Satoshi Nakamoto (whoever he, she or they are) made to the field of computer science*."

# Blockchains (con't)



# Blockchains (con't)

- Other uses for blockchains
- Ethereum
  - Unlike bitcoin, Ethereum uses miniprograms (called smart contracts) that can be written with unlimited complexity
  - Users can then interact with the miniprograms by sending them transactions loaded with instructions, which miners then process
  - What this means is that anyone can embed a software program into a transaction and know that it will remain there, unaltered and accessible for the life span of the blockchain

# Blockchains (con't)

- In theory, Ethereum could replace
  - Facebook, Twitter, Uber, Spotify or any other digital service, with new versions that would be invulnerable to censors and with high integrity
- Another use: Initial Coin Offering (ICO) - think application specific coins, like tokens for a laundromat

# Blockchains (con't)

- Downsides:
  - Computing power consumption
  - Privacy laws
  - Each country has specific privacy laws: financial institutions, medical records

# Blockchains (con't)

# US Security Specifications

- NIST (National Institute of Standards and Technology), see: https://www.nist.gov
- FIPS (Federal Information Processing Standards) and SP800s (Special Publications), see Computer Security Resource Center: https://csrc.nist.gov
- A recent addition is SP800-193, Platform Firmware Resiliency Guidelines, protect against:
  - Unauthorized changes
  - Detecting unauthorized changes
  - Recovery from attacks
- Microsoft has an initiative as well: https://www.microsoft.com/en-us/research/publication/cyber-resilient-platforms-overview/

# Summary

- Develop a Security Mindset, apply “orthogonal” thinking
- Be clear about what are you trying to protect
    - Information/Communication
        - Use encryption
    - Authenticity / Authentication
        - Use MACs
    - Integrity
        - Use Hashes
- Use the current algorithms that are thought to be “secure” at the time
- Address security at all levels and all interfaces in a system
- Always authenticate software/firmware updates - build this into your systems from day one
- Key management is the hard part, but also where you can be creative
- Security is only ever “good enough”
- Security is a never-ending game of cat and mouse
- For more in-depth learning take Eric Wustrow’s course “*Introduction to Computer Security*”

# End