CS 6963/5963
University of Utah

Cyber-physical Systems and IoT Security

Module 1d: Security in a Nutshell (cont'd)
+
Module 1e: Embedded Systems Security (Attacks on Control Flow)

Announcements

- Presentations start next class!
  - First up: "Detecting and Handling IoT Interaction Threats in Multi-Platform Multi-Control-Channel Smart Homes" by RJ Davidson
- Quizzes start next class:
  - Will be released at the beginning of class
  - Once you start, you'll have 5 minutes to complete it.
  - I'm going to start lecturing at the 10-minute mark
- Remember to submit your team members by tonight
  - If you don't have a team, submit an answer indicating:
    - "I don't have a team but would like to be matched; here are my interests"
    - "I don't have a team and would like to work alone
- Schedule a time to chat with me about project ideas if you're struggling with brainstorming

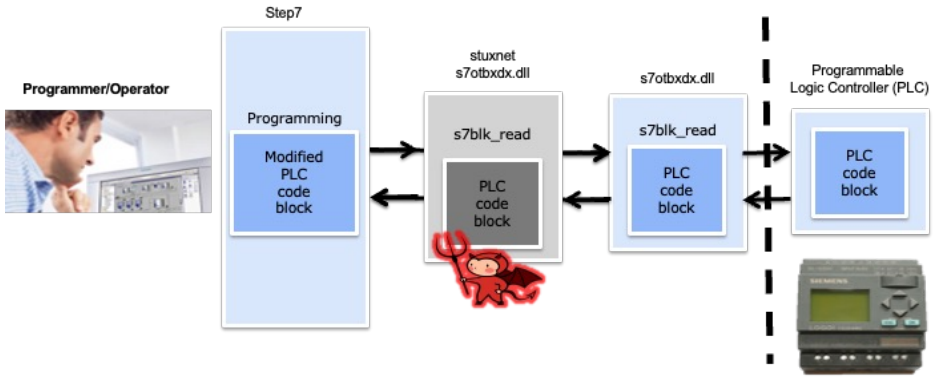

Questions?

Recall: Information System Security Properties

Goal	What does it mean?	How to achieve it?
Confidentiality	Keep your secrets secret	Physical isolation, cryptography, background checks on people
Integrity	Prevent others from modifying your code/data	TPM, Checksums & digital signatures, redundancy, backups
Availability	Make sure you can use your system	Hardening, redundancy, reference checks on people
Authentication	Person or entity is really who it claims to be	Cryptography with public key infrastructure
Control	Regulate the use of your system	Access control lists, physical security
Audit	What happened? How do we undo it?	Log files, provenance info, human auditors, expert systems
Nonrepudiation	Sender / Receiver can't deny sending / receiving	Cryptography with public key infrastructure

Recall: Information System Security

Goal	What does it mean?
Confidentiality	Keep your secrets secret
Integrity	Prevent others from modifying your code/data
Availability	Make sure you can use your system
Authentication	Person or entity is really who it claims to be
Control	Regulate the use of your system
Audit	What happened? How do we undo it?
Nonrepudiation	Sender / Receiver can't deny sending / receiving



Stuxnet: What did the attacker compromise?

Arguably everything on the list

Recall: Information System Security

Goal	What does it mean?
Confidentiality	Keep your secrets secret
Integrity	Prevent others from modifying your code/data
Availability	Make sure you can use your system
Authentication	Person or entity is really who it claims to be
Control	Regulate the use of your system
Audit	What happened? How do we undo it?
Nonrepudiation	Sender / Receiver can't deny sending / receiving

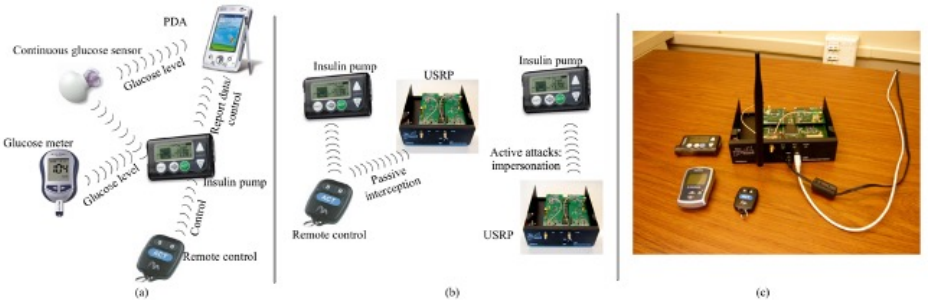


Fig. 1. (a) Insulin delivery system, (b) security attacks, and (c) experimental setup used in the attacks

Attack on Insulin Pump: What did the attacker compromise?

Arguably everything on the list

Recall: Information System Security

Goal	What does it mean?	How to achieve it?
Confidentiality	Keep your secrets secret	Physical isolation, cryptography, background checks on people
Integrity	Prevent others from modifying your code/data	TPM, Checksums & digital signatures, redundancy, backups
Availability	Make sure you can use your system	Hardening, redundancy, reference checks on people
Authentication	Person or entity is really who it claims to be	Cryptography with public key infrastructure
Control	Regulate the use of your system	Access control lists, physical security
Audit	What happened? How do we undo it?	Log files, provenance info, human auditors, expert systems
Nonrepudiation	Sender / Receiver can't deny sending / receiving	Cryptography with public key infrastructure

CPS/IoT Security Research is often at the *application* of Cryptography

- **Goals of this module:**
  - Refresh the very basics of cryptography
  - Set the stage for its application to the world of CPS/IoT

- Crypto is a great **tool** and the basis of many **security mechanisms**, but it's not:
  - The solution to *all* security problems
  - Reliable unless implemented properly
  - Reliable unless used properly

- **Kerchoff's Principle:**
  - A cryptosystem should be secure even if everything about the system, except the secret key, is public knowledge

For a Deeper Dive into Crypto: Take Modern Cryptography (CS5961/CS6961)

- Taught by Pratik Soni
- Deep-dive into various crypto concepts
  - Zero-knowledge proofs
  - Blockchains
  - Homomorphic encryption
  - ....

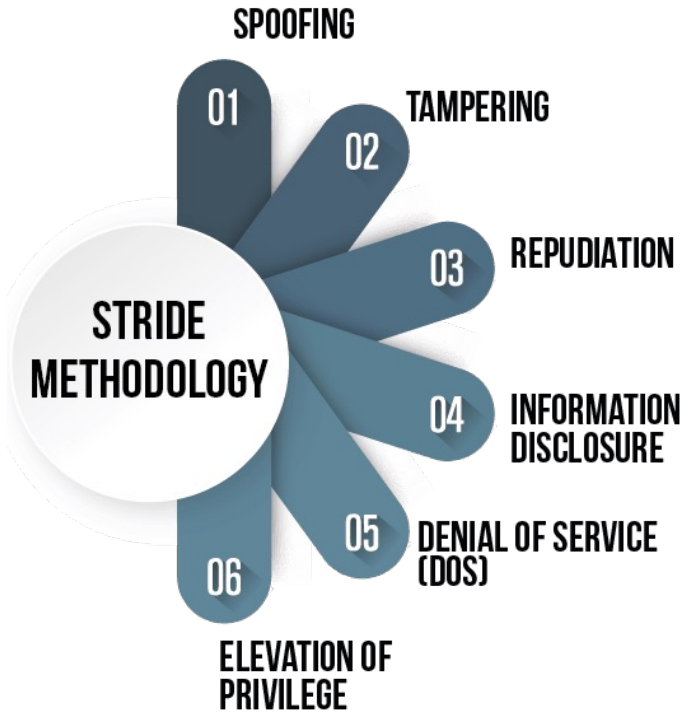
Basic Types of Cryptographic Functions

- **One-way hash functions:** no key
  - Usually used for **integrity**
  - Plaintext ⇨ fingerprint
- **Symmetric crypto:** one key
  - Usually used for **confidentiality**
  - Both Alice and Bob know the same *shared key*
  - Plaintext ⇨ ciphertext
- **Asymmetric Crypto:** two keys
  - Usually used for **confidentiality** and **authenticity**
  - Alice has *public key* and *private key*
  - Everyone knows Alice's *public key*, but only Alice knows her *private key*
  - Encrypt with one and decrypt with the other
  - Plaintext ⇨ ciphertext

Some Basic Attacks on Cryptography

- **Eve and Mallory are the Enemies**
  - Eve can only observe messages
  - Mallory can insert or modify messages
- **For one-way hash**
  - Alter the message without altering its fingerprint
  - Brute-force attack
- **For symmetric/asymmetric crypto**
  - Either learn the keys or learn plaintext from ciphertext
  - Ciphertext-only attack, known-plaintext attack, chose-plaintext attack
  - Main-in-the-middle attack
  - Brute-force attack

In general, lots of different ways to model threats...



Mitre ATT&CK

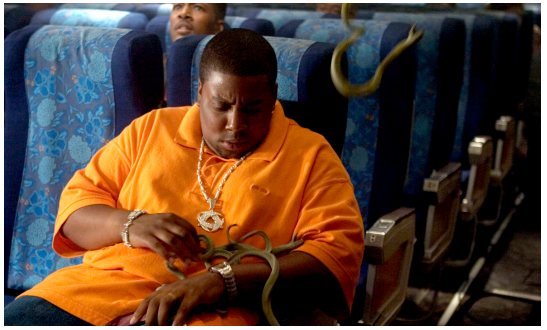
Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Execution	Collection	Exfiltration
51 items	27 items	49 items	18 items	17 items	17 items	25 items	13 items	9 items
.bash_profile and .bashrc	Access Token Manipulation	Access Token Manipulation	Account Manipulation	Account Discovery	AppleScript	AppleScript	Audio Capture	Automated Exfiltratior
Accessibility Features T1015	Accessibility Features	Binary Padding	Bash History	Application Window Discovery	Application Deployment Software	Command-Line Interface	Automated Collection	Data Compressed
AppCert DLLs	AppCert DLLs	Bypass User Account Control	Brute Force	File and Directory Discovery	Distributed Component Object Model	Dynamic Data Exchange	Browser Extensions	Data Encrypted
AppInit DLLs	AppInit DLLs	Clear Command History	Credential Dumping	Network Service Scanning	Exploitation of Vulnerability	Execution through API	Clipboard Data	Data Transfer Size Limits
Application Shimming	Application Shimming	Code Signing	Credentials in Files	Network Share Discovery	Logon Scripts	Execution through Module Load	Data from Local System	Exfiltration Over Alternative Protocol
Authentication Package	Bypass User Account Control	Component Firmware	Exploitation of Vulnerability	Peripheral Device Discovery	Pass the Hash	Graphical User Interface	Data from Network Shared Drive	Exfiltration Over Command and Contro Channel
Bootkit	DLL Search Order Hijacking	Component Object Model Hijacking	Forced Authentication	Permission Groups Discovery	Pass the Ticket	InstallUtil	Data from Removable Media	Exfiltration Over Othe Network Medium
Browser Extensions	Dylib Hijacking	Deobfuscate/Decode Files or Information	Hooking	Process Discovery	Remote Desktop Protocol	Launchctl	Data Staged	Exfiltration Over Physical Medium
Change Default File Association	Exploitation of Vulnerability	Disabling Security Tools	Input Capture	Query Registry	Remote File Copy	Local Job Scheduling	Email Collection	Scheduled Transfer
Component Firmware	Extra Window Memory Injection	DLL Search Order Hijacking	Input Prompt	Remote System Discovery	Remote Services	LSASS Driver	Input Capture	
Component Object Model Hijacking	File System Permissions Weakness	DLL Side-Loading	Keychain	Security Software Discovery	Replication Through Removable Media	Mshta	Man in the Browser	
Create Account	Hooking	Exploitation of Vulnerability	LLMNR/NBT-NS Poisoning	System Information Discovery	Shared Webroot	PowerShell	Screen Capture	
DLL Search Order Hijacking	Image File Execution Options Injection	Extra Window Memory Injection	Network Sniffing	System Network Configuration Discovery	SSH Hijacking	Regsvcs/Regasm	Video Capture	
Dylib Hijacking	Launch Daemon	File Deletion	Password Filter DLL	System Network Connections Discovery	Taint Shared Content	Regsvr32		
External Remote Services	New Service	File System Logical Offsets	Private Keys	System Owner/User Discovery	Third-party Software	Rundll32		
File System Permissions Weakness	Path Interception	Gatekeeper Bypass	Replication Through Removable Media		Windows Admin Shares	Scheduled Task		
Hidden Files and Directories	Plist Modification	Hidden Files and Directories	Securityd Memory		Windows Remote Management	Scripting		
Hooking	Port Monitors	Hidden Users	Two-Factor Authentication Interception			Service Execution		
Hypervisor		Hidden Window				Source		
Image File Execution		HISTCONTROL				Space after Filename		
		Image File Execution Options				Third-party Software		

Modeling Threats:
Common Elements You'll Find in Security Research Papers

Research papers typically have a different structure to identify, quantify, and address security risks associated with an application

Common Paper Components (I'll expect these in your projects eventually):

1. **System Model: sets the stage for what vulns might exist**
   - **Components:** What are the parts components of the systems?
   - **Interactions:** How do these parts communicate or interact?
   - **Behaviors:** What is the expected behavior of these components
2. **Adversary or Threat Model:**
   - **Capabilities:** What can the adversary do?
   - **Knowledge:** What does the adversary know?
   - **Goals:** What is the adversary trying to achieve?
3. **Assumptions:** Clearly state any assumptions made either about the system or adversary, users, etc.
4. **Out of Scope:** Clearly defined what is not being addressed to manage expectations
   - "we can protect the airplane against GPS spoofing, but we don't protect against snakes!"



One-way Hash Functions

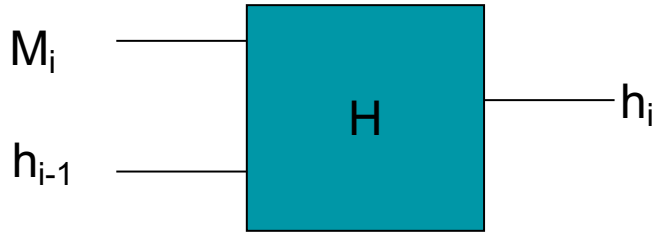
- Take a variable-length input *M* and produce fixed-length output (*hash value* or *message digest)*

$$h = H(M)$$

- M can be a file, a message, or anything you are trying to produce a hash
- One-way hash function is public
- One-way hash function requirements:
  1. M can be any size
  2. h is always fixed size
  3. Given M it is very easy/fast to produce h
  4. Given *h it is* very hard to compute *M* (one-way property) → non-reversible
  5. Given *M* it is very hard to find M ′ such that H(M)=H(M′) (weak collision resistance)
  6. It is hard to find two random messages M_1 and M_2 such that H(M_1)=H(M_2) (strong collision resistance)

One-Way Hash Functions: Large Messages

- Divide M into blocks, generate hash value iteratively



- Hash value of the whole message is obtained in the last step
- For a hash function that produces *m* bit hash, it takes $2^{m/2}$ trials to find any two messages that hash to the same value by just trying every possible message
  - We need large *m*, currently 256-512

Uses Of Hash Algorithms

- Storing hashed password info
  - In Linux, hashed passwords stored in /etc/shadow
- Message integrity
  - Use message M and a shared secret S, run this through hash function and produce MIC = secure hash
  - Send only M and MIC
  - Is confidentiality protected?
  - Why do we need a shared secret?
- Message fingerprint
  - Hash the files to detect tampering
  - Works for download security too
- Signing message hash instead of the whole message is faster

Uses Of Hash Algorithms

- Storing hashed password info
  - In Linux, hashed passwords stored in /etc/shadow
- Messag
  - Use m
    run thi
  - Send
  - Is con
  - Why d
- Messag
  - Hash
  - Works
- Signing

JDK 20 JDK 17 GraalVM for JDK 20 GraalVM for JDK 17 https://www.oracle.com/java/technologies/downloads/

JDK Development Kit 20.0.2 downloads

JDK 20 binaries are free to use in production and free to redistribute, at no cost, under the Oracle No-Fee Terms and Conditions.

JDK 20 will receive updates under these terms, until September 2023 when it will be superseded by JDK 21.

Linux macOS Windows

Product/file description	File size	Download
ARM64 Compressed Archive	181.55 MB	https://download.oracle.com/java/20/latest/jdk-20_linux-aarch64_bin.tar.gz (sha256)
ARM64 RPM Package	181.27 MB	https://download.oracle.com/java/20/latest/jdk-20_linux-aarch64_bin.rpm (sha256) (OL 8 GPG Key)
x64 Compressed Archive	183.11 MB	https://download.oracle.com/java/20/latest/jdk-20_linux-x64_bin.tar.gz (sha256)
x64 Debian Package	155.91 MB	https://download.oracle.com/java/20/latest/jdk-20_linux-x64_bin.deb (sha256)
x64 RPM Package	182.82 MB	https://download.oracle.com/java/20/latest/jdk-20_linux-x64_bin.rpm (sha256) (OL 8 GPG Key)

Some Standard Hash Algorithms

- MD5 – 1991
  - Some weaknesses discovered, not secure
- SHA-1 – 1993
  - Some weaknesses discovered
  - Revised in 1995
  - 160 bits
- SHA-2 - 2002
  - Still using pieces of SHA-1 algorithm
  - 256, 384 and 512 bits
  - Most widely used today
- SHA-3 – 2015
  - A completely different algorithm

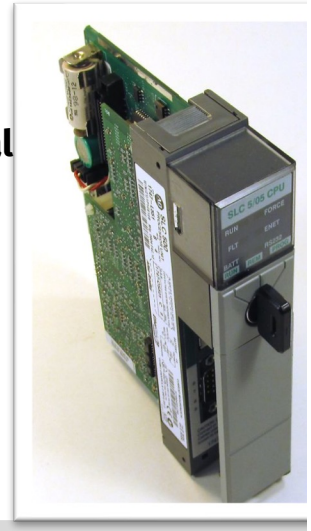
Previous Attack on Integrity: PLC Firmware Modification

- Targeted checksum mechanism (last 8 bytes)
  - Found CRC algo used and simply replaced last 8 bytes with updated value
- Next generation:
  - Send firmware with **digital signature**

3-21-2013

Firmware Counterfeiting and Modification Attacks on Programmable Logic Controllers

Zachary H. Basnight



```
FRN16.081\PN-66834.bin
001D EB50: 74 65 6D 73 5C 4C 78 5C  4C 78 53 61 66 65 74 79  tems\Lx\ LxSafety
001D EB60: 44 75 61 6C 4C 69 73 74  2E 68 70 70 00 00 00 00  DualList .hpp....
001D EB70: 2E 2E 5C 2E 2E 5C 53 6F  75 72 63 65 5C 61 6C 6D  ..\..\So urce\alm
001D EB80: 66 6E 63 2E 68 00 00 00  2E 2E 5C 2E 2E 5C 53 6F  fnc.h... ..\..\So
001D EB90: 75 72 63 65 5C 61 6C 6D  66 6E 63 2E 68 00 00 00  urce\alm fnc.h...
001D EBA0: 00 00 F0 7F 00 00 00 00  00 00 F8 7F 00 00 00 00  ..≡⌂.... ..°⌂....
001D EBB0: 00 00 80 7F 00 00 C0 7F  00 00 D0 00 E8 EB ED 00  ..Ç⌂..└⌂ ..╨.ΦδΦ.
001D EBC0: 00 00 00 00 00 00 00 00  00 00 00 00 E8 EB ED 00  ........ ....ΦδΦ.
001D EBD0: 00 00 00 60 54 00 00 60  40 18 00 00 90 9C 03 00  ...`T..` @...éÉ..
001D EBE0: 80 00 00 0C 94 36 00 0C  03 00 A0 E3 10 0F 07 EE  Ç....ö6.. ..áΠ...€
001D EBF0: 10 0F 03 EE 50 14 A0 E3  00 00 A0 E3 00 00 81 E5  ...€P.áΠ ..áΠ..üσ
001D EC00: 2C 10 9F E5 FF 00 A0 E3  00 00 81 E5 D1 00 A0 E3  ,.fσ .áΠ ..üσ╤.áΠ
001D EC10: 00 00 81 E5 21 00 A0 E3  00 00 81 E5 14 10 9F E5  ..üσ!.áΠ ..üσ..fσ
001D EC20: 00 20 91 E5 40 20 C2 E3  40 20 82 E3 00 20 81 E5  . æσ@ ┬Π @ éΠ. üσ
001D EC30: FE FF FF EA 8C 05 01 08  80 02 01 08 70 D3 49 4D  ■  ΩÎ... Ç...pᵘIM
001D EC40: 78 2A 8F C4                                      x*ÅR─
001D EC50:
FRN16.057\PN-66830.bin
001D EB50: 74 65 6D 73 5C 4C 78 5C  4C 78 53 61 66 65 74 79  tems\Lx\ LxSafety
001D EB60: 44 75 61 6C 4C 69 73 74  2E 68 70 70 00 00 00 00  DualList .hpp....
001D EB70: 2E 2E 5C 2E 2E 5C 53 6F  75 72 63 65 5C 61 6C 6D  ..\..\So urce\alm
001D EB80: 66 6E 63 2E 68 00 00 00  2E 2E 5C 2E 2E 5C 53 6F  fnc.h... ..\..\So
001D EB90: 75 72 63 65 5C 61 6C 6D  66 6E 63 2E 68 00 00 00  urce\alm fnc.h...
001D EBA0: 00 00 F0 7F 00 00 00 00  00 00 F8 7F 00 00 00 00  ..≡⌂.... ..°⌂....
001D EBB0: 00 00 80 7F 00 00 C0 7F  00 00 D0 00 E8 EB ED 00  ..Ç⌂..└⌂ ..╨.ΦδΦ.
001D EBC0: 00 00 00 00 00 00 00 00  00 00 00 00 E8 EB ED 00  ........ ....ΦδΦ.
001D EBD0: 00 00 00 60 54 00 00 60  40 18 00 00 90 9C 03 00  ...`T..` @...éÉ..
001D EBE0: 80 00 00 0C 94 36 00 0C  03 00 A0 E3 10 0F 07 EE  Ç....ö6.. ..áΠ...€
001D EBF0: 10 0F 03 EE 50 14 A0 E3  00 00 A0 E3 00 00 81 E5  ...€P.áΠ ..áΠ..üσ
001D EC00: 2C 10 9F E5 FF 00 A0 E3  00 00 81 E5 D1 00 A0 E3  ,.fσ .áΠ ..üσ╤.áΠ
001D EC10: 00 00 81 E5 21 00 A0 E3  00 00 81 E5 14 10 9F E5  ..üσ!.áΠ ..üσ..fσ
001D EC20: 00 20 91 E5 40 20 C2 E3  40 20 82 E3 00 20 81 E5  . æσ@ ┬Π @ éΠ. üσ
001D EC30: FE FF FF EA 8C 05 01 08  80 02 01 08 13 23 39 4C  ■  ΩÎ... Ç....#9L
001D EC40: 19 F2 9F C5                                      .≥fƒ
001D EC50:

Arrow keys move   F find        RET next difference   ESC quit   ALT  freeze top
C ASCII/EBCDIC    E edit file     G goto position       Q quit     CTRL freeze bottom
```

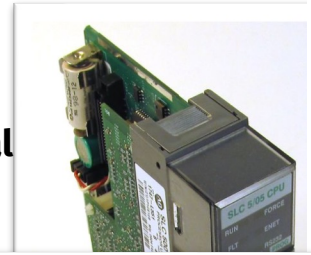
Previous Attack on Integrity: PLC Firmware Modification

- Targeted checksum mechanism (last 8 bytes)
  - Found CRC algo used and simply replaced last 8 bytes with updated value
- Next generation:
  - Send firmware with **digital signature**

3-21-2013

Firmware Counterfeiting and Modification Attacks on Programmable Logic Controllers

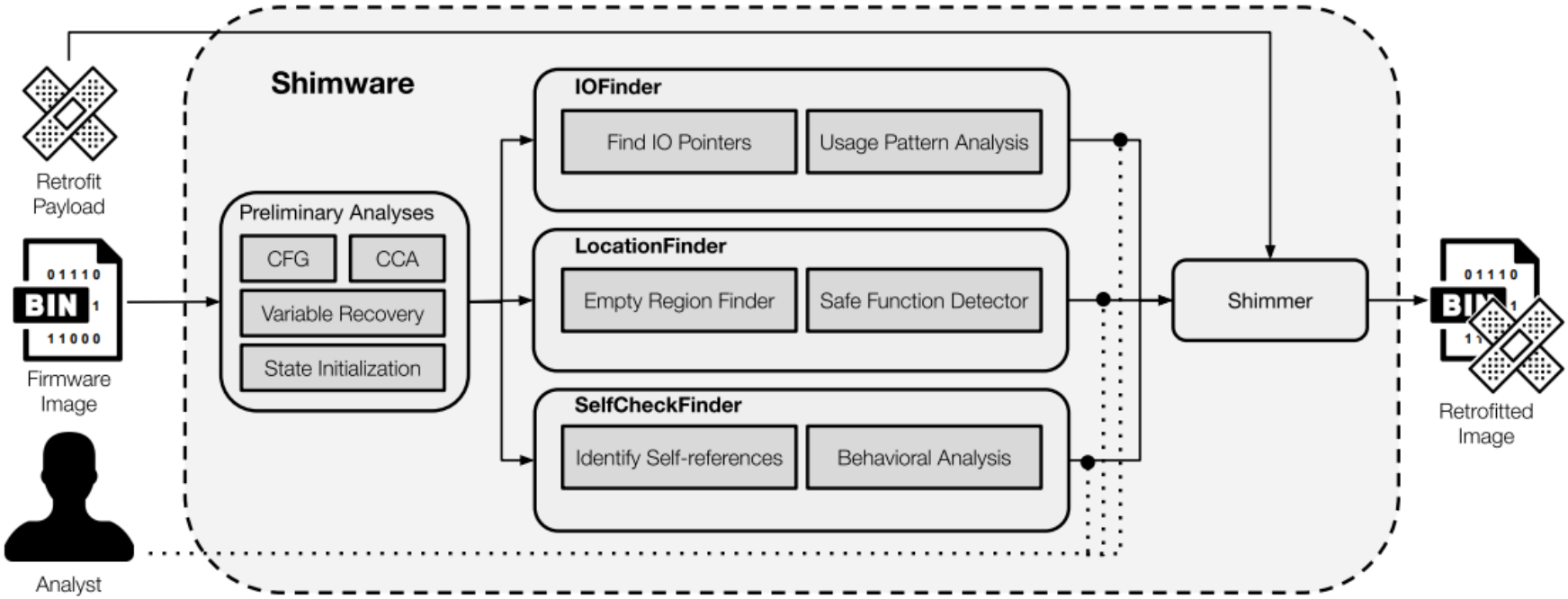
Zachary H. Basnight



```
FRN16.081\PN-66834.bin
001D EB50: 74 65 6D 73 5C 4C 78 5C  4C 78 53 61 66 65 74 79  tems\Lx\ LxSafety
001D EB60: 44 75 61 6C 4C 69 73 74  2E 68 70 70 00 00 00 00  DualList .hpp....
001D EB70: 2E 2E 5C 2E 2E 5C 53 6F  75 72 63 65 5C 61 6C 6D  ..\..\So urce\alm
001D EB80: 66 6E 63 2E 68 00 00 00  2E 2E 5C 2E 2E 5C 53 6F  fnc.h... ..\..\So
001D EB90: 75 72 63 65 5C 61 6C 6D  66 6E 63 2E 68 00 00 00  urce\alm fnc.h...
001D EBA0: 00 00 F0 7F 00 00 00 00  00 00 F8 7F 00 00 00 00  ..≡⌂.... ..°⌂....
001D EBB0: 00 00 80 7F 00 00 C0 7F  00 00 D0 00 E8 EB ED 00  ..Ç⌂..└⌂ ..╨.ΦδΦ.
001D EBC0: 00 00 00 00 00 00 00 00  00 00 00 00 E8 EB ED 00  ........ ....ΦδΦ.
001D EBD0: 00 00 00 60 54 00 00 60  40 18 00 00 90 9C 03 00  ...`T..` @...éÉ..
001D EBE0: 80 00 00 0C 94 36 00 0C  03 00 A0 E3 10 0F 07 EE  Ç....ö6.. ..áΠ...€
001D EBF0: 10 0F 03 EE 50 14 A0 E3  00 00 A0 E3 00 00 81 E5  ...€P.áΠ ..áΠ..üσ
001D EC00: 2C 10 9F E5 FF 00 A0 E3  00 00 81 E5 D1 00 A0 E3  ,.fσ .áΠ ..üσ╤.áΠ
001D EC10: 00 00 81 E5 21 00 A0 E3  00 00 81 E5 14 10 9F E5  ..üσ!.áΠ ..üσ..fσ
001D EC20: 00 20 91 E5 40 20 C2 E3  40 20 82 E3 00 20 81 E5  . æσ@ ┬Π @ éΠ. üσ
001D EC30: FE FF FF EA 8C 05 01 08  80 02 01 08 70 D3 49 4D  ■  ΩÎ... Ç...pᵘIM
001D EC40: 78 2A 8F C4                                      x*Å─
001D EC50:
FRN16.057\PN-66830.bin
001D EB50: 74 65 6D 73 5C 4C 78 5C  4C 78 53 61 66 65 74 79  tems\Lx\ LxSafety
001D EB60: 44 75 61 6C 4C 69 73 74  2E 68 70 70 00 00 00 00  DualList .hpp....
001D EB70: 2E 2E 5C 2E 2E 5C 53 6F  75 72 63 65 5C 61 6C 6D  ..\..\So urce\alm
001D EB80: 66 6E 63 2E 68 00 00 00  2E 2E 5C 2E 2E 5C 53 6F  fnc.h... ..\..\So
001D EB90: 75 72 63 65 5C 61 6C 6D  66 6E 63 2E 68 00 00 00  urce\alm fnc.h...
001D EBA0: 00 00 F0 7F 00 00 00 00  00 00 F8 7F 00 00 00 00  ..≡⌂.... ..°⌂....
001D EBB0: 00 00 80 7F 00 00 C0 7F  00 00 D0 00 E8 EB ED 00  ..Ç⌂..└⌂ ..╨.ΦδΦ.
001D EBC0: 00 00 00 00 00 00 00 00  00 00 00 00 E8 EB ED 00  ........ ....ΦδΦ.
001D EBD0: 00 00 00 60 54 00 00 60  40 18 00 00 90 9C 03 00  ...`T..` @...éÉ..
001D EBE0: 80 00 00 0C 94 36 00 0C  03 00 A0 E3 10 0F 07 EE  Ç....ö6.. ..áΠ...€
001D EBF0: 10 0F 03 EE 50 14 A0 E3  00 00 A0 E3 00 00 81 E5  ...€P.áΠ ..áΠ..üσ
001D EC00: 2C 10 9F E5 FF 00 A0 E3  00 00 81 E5 D1 00 A0 E3  ,.fσ .áΠ ..üσ╤.áΠ
001D EC10: 00 00 81 E5 21 00 A0 E3  00 00 81 E5 14 10 9F E5  ..üσ!.áΠ ..üσ..fσ
001D EC20: 00 20 91 E5 40 20 C2 E3  40 20 82 E3 00 20 81 E5  . æσ@ ┬Π @ éΠ. üσ
001D EC30: FE FF FF EA 8C 05 01 08  80 02 01 08 13 23 39 4C  ■  ΩÎ... Ç....#9L
001D EC40: 19 F2 9F C5                                      ↓≥ƒ┼
001D EC50:
Arrow keys move  F find        RET next difference  ESC quit  ALT  freeze top
C ASCII/EBCDIC   E edit file     G goto position      Q quit    CTRL freeze bottom
```

PN-311149.nvs	4/28/2015 1:41 PM	NVS File	7 KB
PN-311149	4/28/2015 3:48 PM	Compiled Resource Script	1 KB
PN-311150	4/24/2015 7:36 AM	VLC media file (.bin)	3,584 KB
PN-311151	4/24/2015 7:37 AM	VLC media file (.bin)	4,480 KB
PN-311152	4/24/2015 7:37 AM	Security Certificate	1 KB
PN-311153	4/24/2015 7:37 AM	Security Certificate	1 KB

Identifying Self-checks for Good: Binary Retrofitting



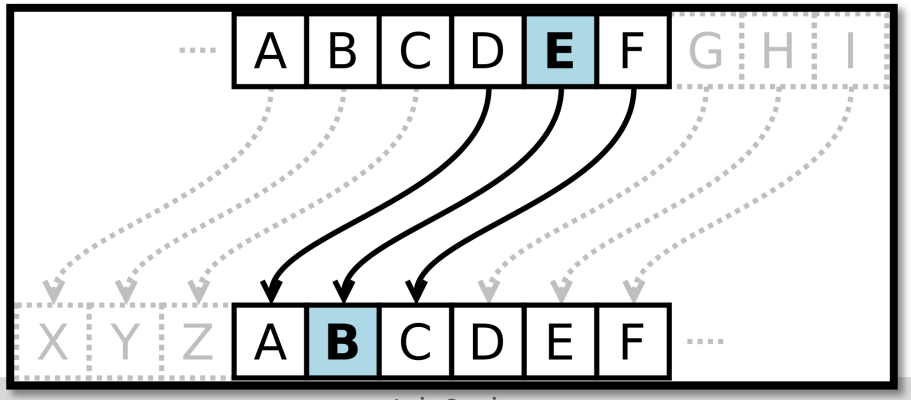
https://sites.cs.ucsb.edu/~vigna/publications/2023_RAID_Shimware.pdf

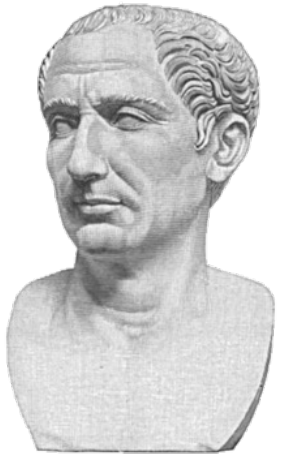
Symmetric Encryption

- Scramble messages between Alice and Bob so others who see the messages cannot make sense of them
- Terminology
  - Message to be sent – **plaintext**
  - Scrambled message – **ciphertext**
  - Scrambling process – plain2cipher - **encryption**
  - Unscrambling process – cipher2plain - **decryption**
  - Key used to encrypt/decrypt – **shared key** or **secret key**
- Reminder: Good cryptography assumes knowledge of the algorithm
  - Secret lies in a key!
- Symmetric Crypto: Alice and Bob use the same key to both encrypt and decrypt messages

Example: Caesar's Cipher

- Substitute each letter with a letter which is 3 letters later in the alphabet
  - HELLO becomes KHOOR
- Instead of using number 3 we could use $n \in [1,25]$. n would be our key
  - Why can't we use n = 26, 0?
- How can we break this cipher? Can you decipher this:
  Bpqa kzgxbwozixpg ammua zmit miag. Em eivb uwzm!





https://www.dcode.fr/caesar-cipher

Symmetric Crypto Algorithms

- Requirements:
  - An opponent who knows the algo and has several ciphertexts cannot decipher them or retrieve the key
  - An opponent who knows the algo and has several ciphertext and plaintext pairs cannot retrieve the key

Symmetric Cryptographic Techniques

- **Substitution**
  - Goal: obscure relationship between plaintext and ciphertext
  - Substitute parts of plaintext with parts of ciphertext
    - **What are some issues that might arise?**

- **Transposition (shuffling)**
  - Goal: dissipate redundancy of the plaintext by spreading it over ciphertext
  - This way changing one bit of plaintext affects many bits of the ciphertext (if we have rounds of encryption)

What was Caesar's Cipher?

Substitution: Monoalphabetic

- Monoalphabetic – each character is replaced with another character
  - Caesar's cipher—each letter is shifted by 3, a becomes d, b becomes e, etc.
  - Keep a mapping of symbols into other symbols
  - **Drawback:** frequency of symbols stays the same and can be used to break the cipher

https://www.101computing.net/frequency-analysis/

Substitution: Homophonic

- Homophonic – each character is replaced with a character chosen randomly from a subset
  - Ciphertext alphabet must be larger than plaintext alphabet – we could replace letters by two-digit numbers
  - Number of symbols in the subset depend on frequency of the given letter in the plaintext
  - The resulting ciphertext has all alphabet symbols appearing with the same frequency

Substitution: Homophonic

- **e** is 4 times as frequent as **b**, and **b** is twice as frequent as **a** and **d**
- **a** – 11, **d** – 22, **b** – 33, 44, **e** – 12, 13, 87, 65, 93, 20, 47, 88
- Example: bebebeabed

  33 87 44 93 44 20 11 33 12 22

Substitution: Polygram

- Polygram – each sequence of characters of length *n* is replaced with another sequence of characters of length *n*
  - Like monoalphabetic cipher but works on *n*-grams
  - Example: bebebeabedxx
  - bebe – uyri, bebb – ssjd, beab – ooiw, edxx – skqi
  - Translation: uyriooiwskqi

Substitution: Polyalphabetic

- Polyalphabetic – many monoalphabetic ciphers are used sequentially
  - First mapping is used for the first letter, second mapping for the second letter and so on
    - Sender and receiver need to know switching frequency and switch at the same time
  - XOR is a polyalphabetic cyber in binary domain

Plaintext: “011010”
Key: “101010”
XOR’d ciphertext: “110000”

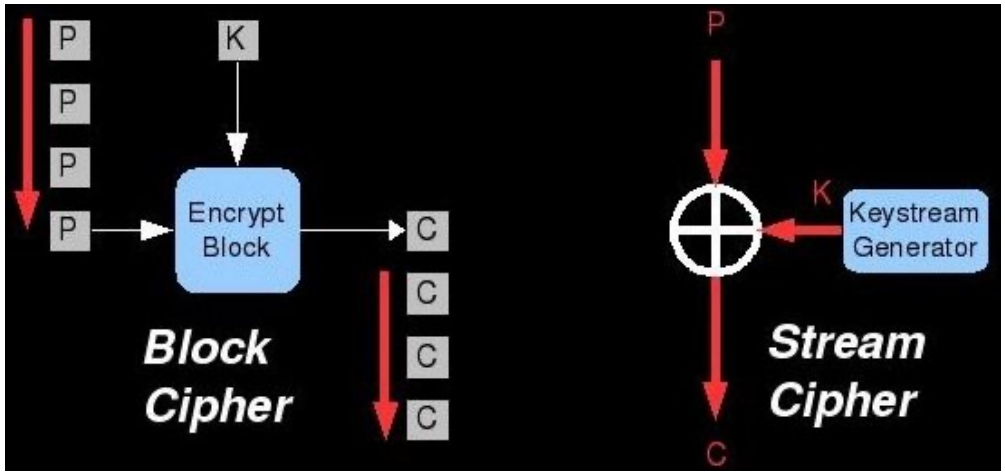
Plaintext: “011010”
Key (shifted by 1): “010101”
XOR’d ciphertext: “001111”

Substitution: Polyalphabetic

- Polyalphabetic – many monoalphabetic ciphers are used sequentially
  - First mapping is used for the first letter, second mapping for the second letter and so on
    - Sender and receiver need to know switching frequency and switch at the same time
  - XOR is a polyalphabetic cyber in binary domain
  - Example: bebebeabed, 2 mappings,
    - b– x, e – y, a – t, d – p
    - b – a, e – x, a – q, d – I
    - xxxxxxtap

Symmetric Crypto Algorithms

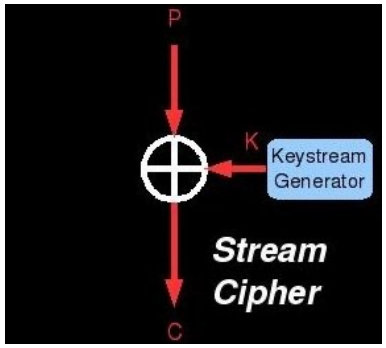
- Typically used in 2 ways



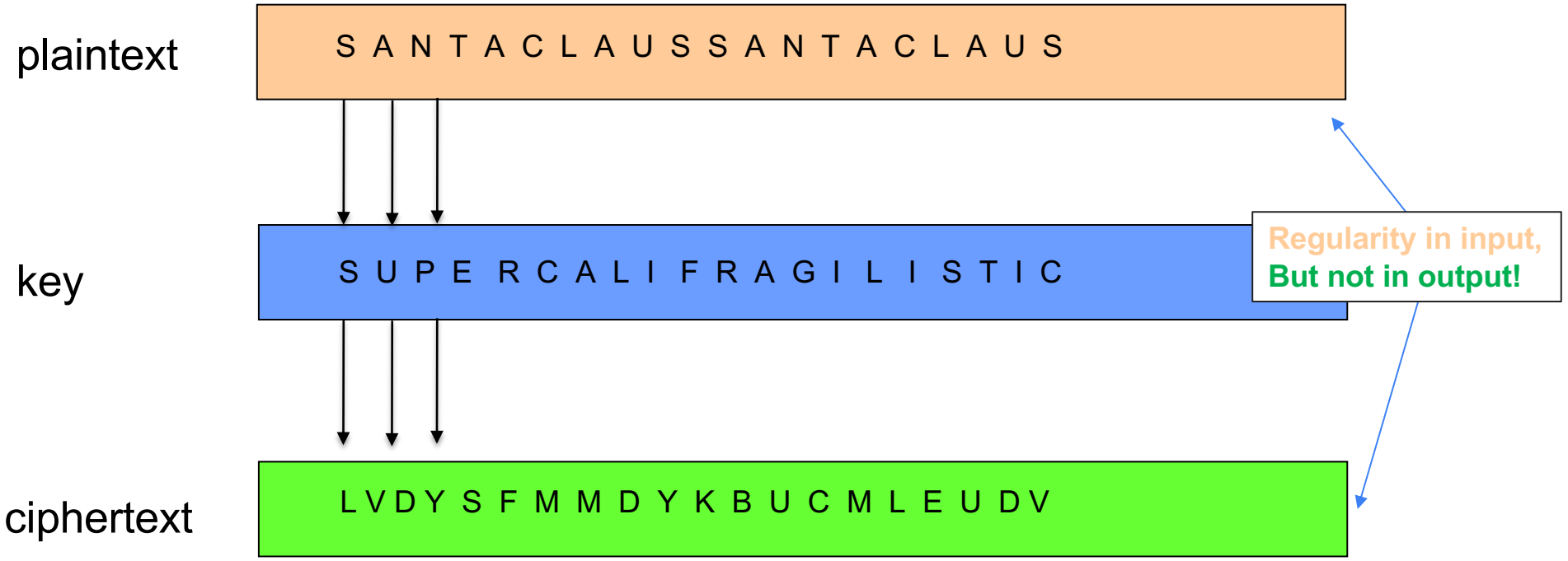
http://theamazingking.com/crypto-basics.php

Symmetric Crypto Algorithms: Stream Ciphers

- Stream ciphers are polyalphabetic
- Work on message a bit or a byte at a time
  - Assume XOR with the key, we have this property:
    - A xor K = B
    - A xor B = K
  - Same bit/byte will encrypt differently, depending on the position of the key
    - Key needs to be very long, random, and unpredictable



Stream Cipher Example

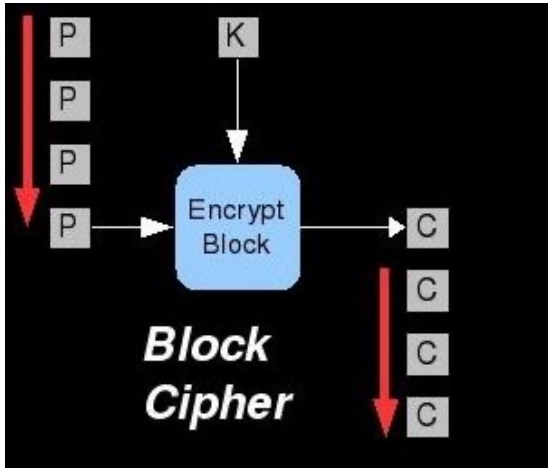


One-Time Pad

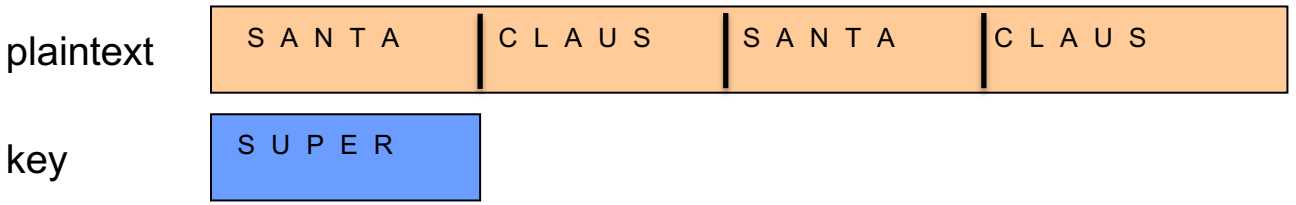
- Polyalphabetic cipher with “infinite” key
  - Combine letters from the message with the letters from an infinite key, randomly generated
  - Never reuse the key
  - Key needs to be generated using a very good RNG (to avoid any patterns)
- This cipher cannot be broken
  - Any info the attacker gains about the key will be useless!
- Sender and receiver must be perfectly synchronized

Symmetric Crypto Algorithms: Block Ciphers

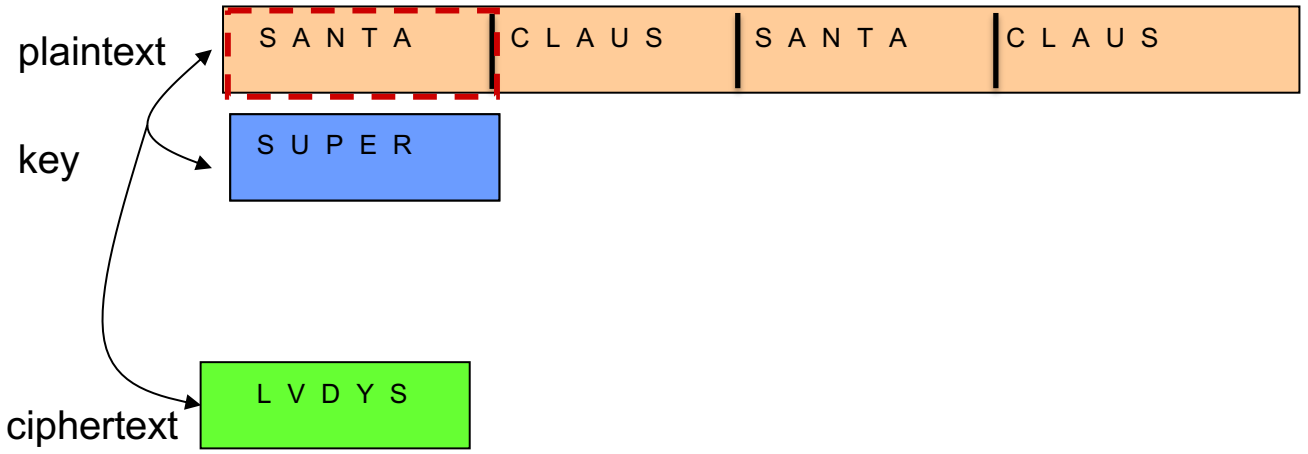
- Block ciphers are polygrams
  - Work message block by block
  - Block size is usually the same as key size
  - Same plaintext block may encrypt into the same ciphertext block, depending on cipher mode



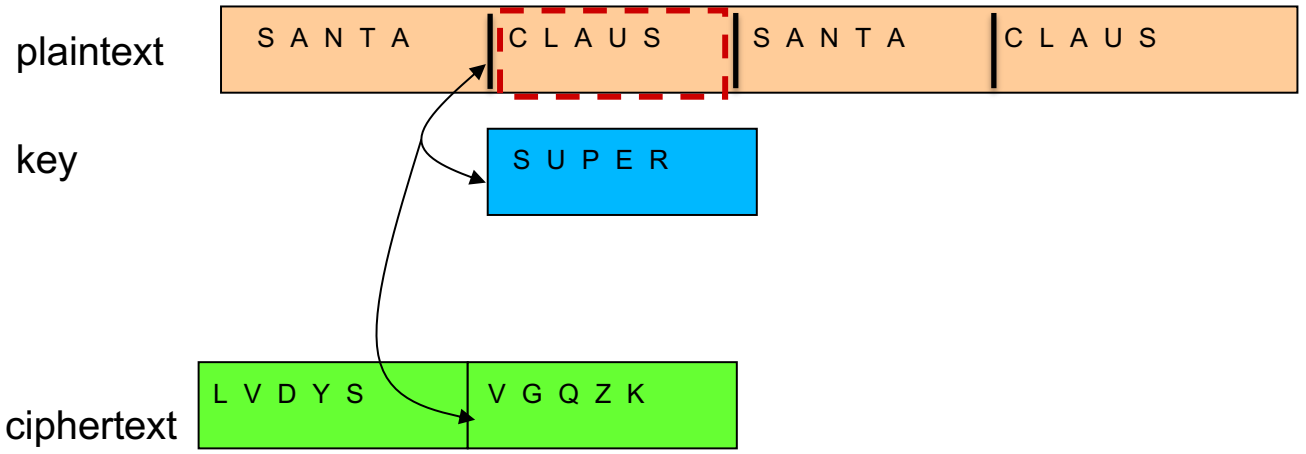
Block Cipher Example



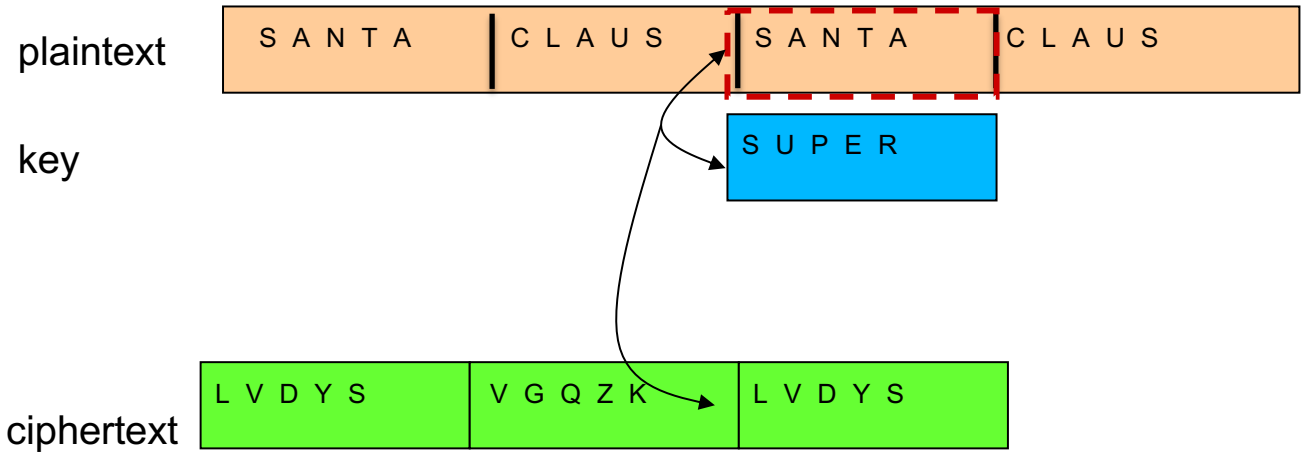
Block Cipher Example



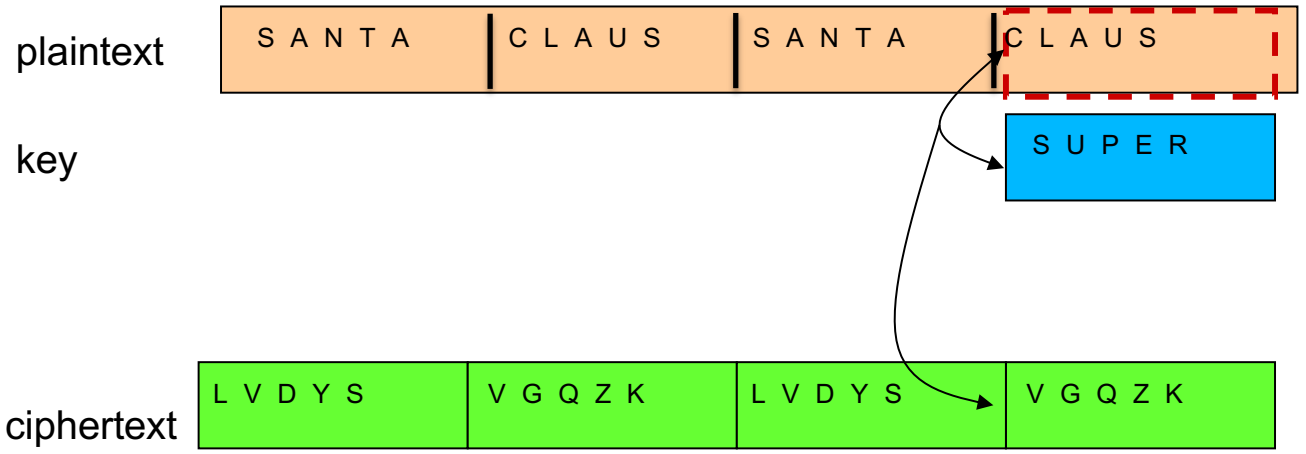
Block Cipher Example



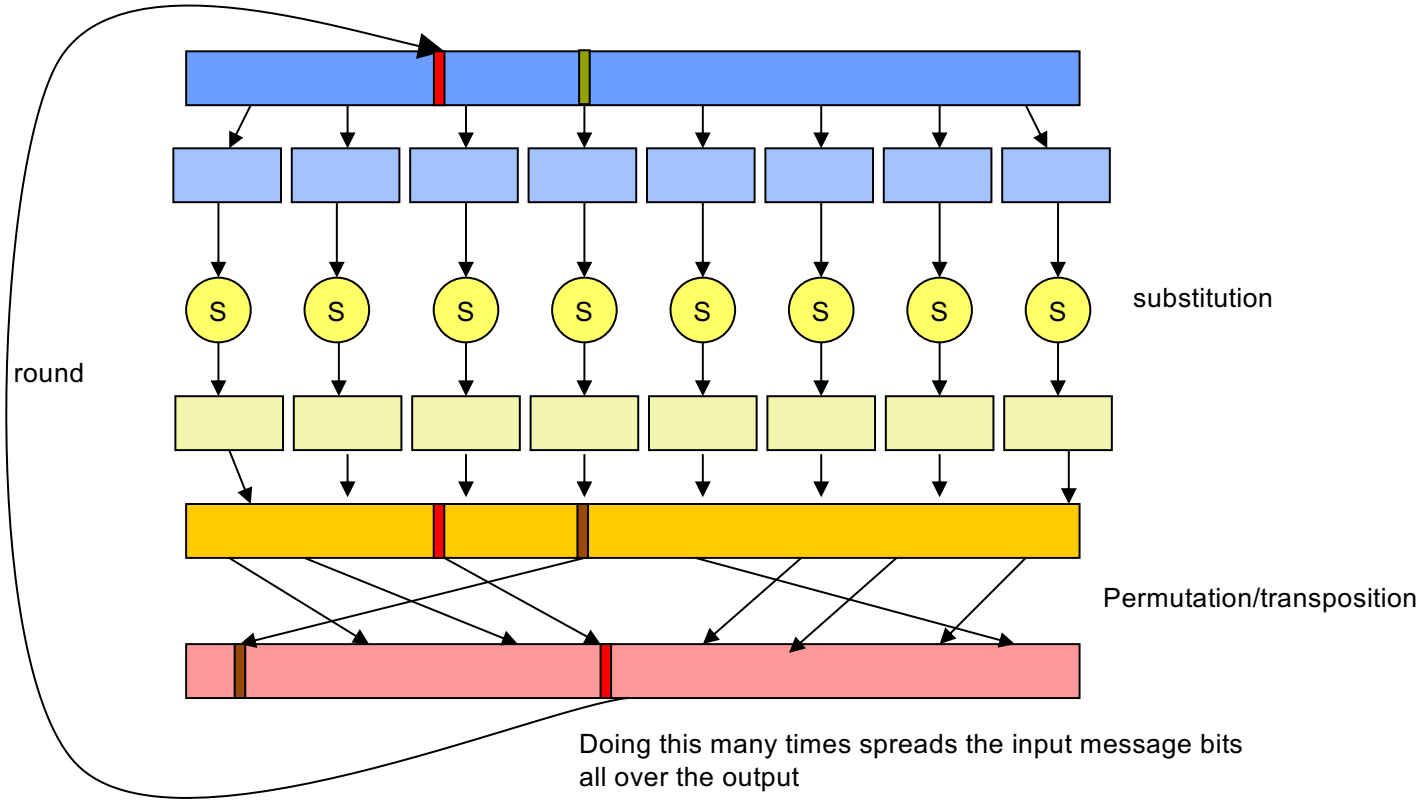
Block Cipher Example



Block Cipher Example



Block Encryption In Rounds



Symmetric Crypto Standards

- Block ciphers
  - DES – Data Encryption Standard – 1977
    - 64 bit block size, 56 bit key size
    - Too small given today's computational power
  - 3-DES
    - Same as DES but each text is processed 3 times with 3 different keys
    - E_{K1}, D_{K2}, E_{K3}
  - AES – Advanced Encryption Standard
    - 128 bit block size, 128, 192 or 256 bit key size
- Stream ciphers
  - RC4
    - From 8 to 2048 bit key

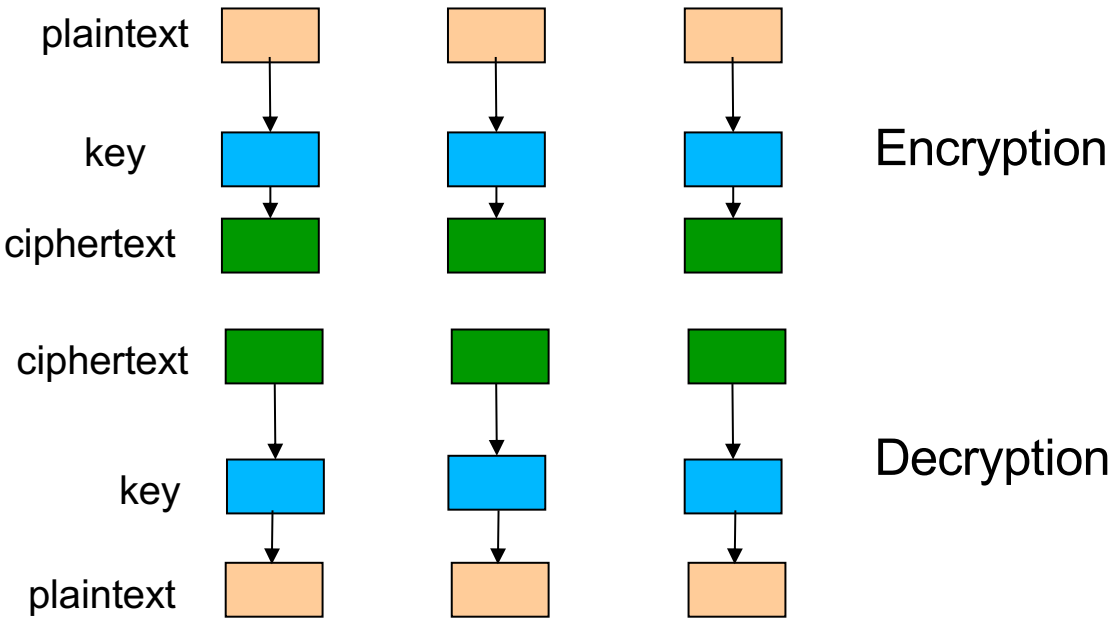
Large Messages

- How do you encrypt a large message using block cypher?
- Break it into smaller pieces and encrypt each piece
  - Decide whether latter pieces may depend on the earlier ones
- Electronic Code Book (ECB)
  - No dependency
- Cipher Block Chaining (CBC)
- A few other modes which we will not cover

Electronic Code Book (ECB)

- Store mapping for every possible block
  - Fast encryption/decryption—just a table lookup
  - Ability to process text in any order and in parallel
  - Table size could be enormous→ make the mapping depend on the key

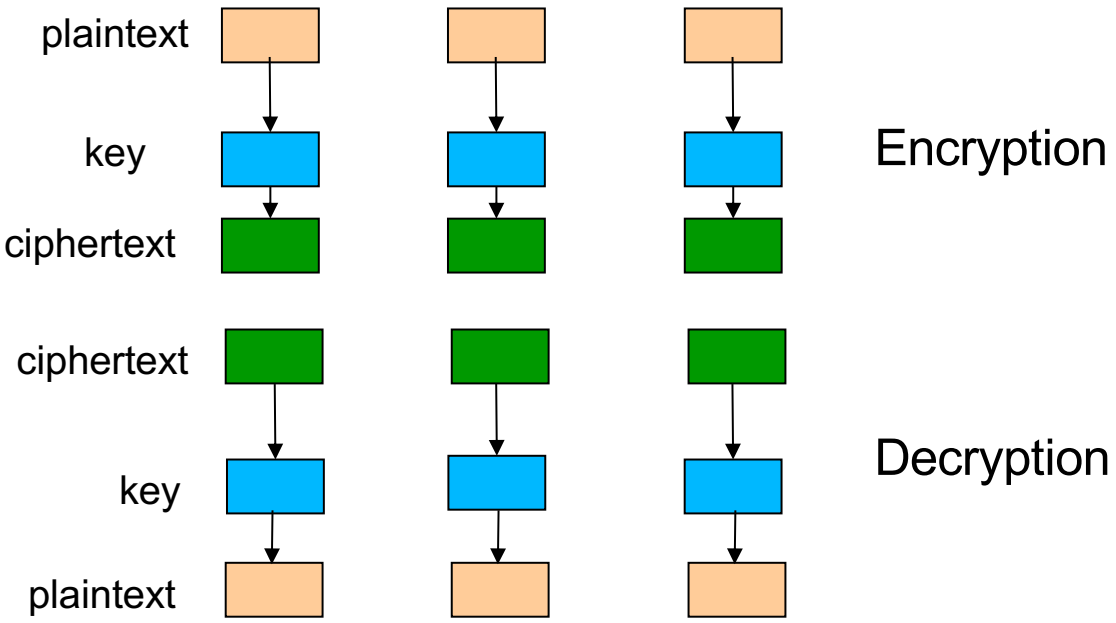
Electronic Code Book (ECB)



ECB Problems

- Mallory can detect which blocks map to other blocks by seeing several plaintext and corresponding ciphertext messages
- Can replay messages
- Can fabricate new messages out of pieces

Electronic Code Book (ECB)



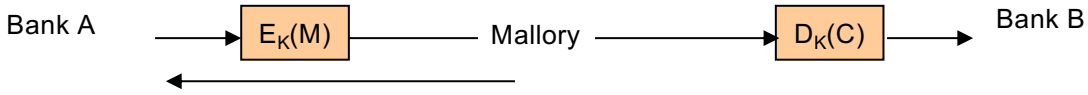
Message Replay

- Alice transfers $100 to Bob's account in Bank B
- Mallory can replay 12B7 7783 1120 8080 at will
  - Replay attack – capture a message, copy it, let it go and later repeat the same message
  - Mallory can be working with Bob or just wants to make Alice poor

Message Fabrication

tstamp transfer amt src dst

3231 12B7 7783 38AC CDC7



- Bank adds timestamps to prevent replay
- Mallory transfers a few amounts, learns the structure:

Transfer 10$ from X to Y	3231 12B7 2212 38AC CDC7
Transfer 5$ from Y to X	8001 12B7 7211 CDC7 38AC
Transfer 5$ from X to Y	7272 12B7 7211 38AC CDC7

tstamp transfer amt src dst

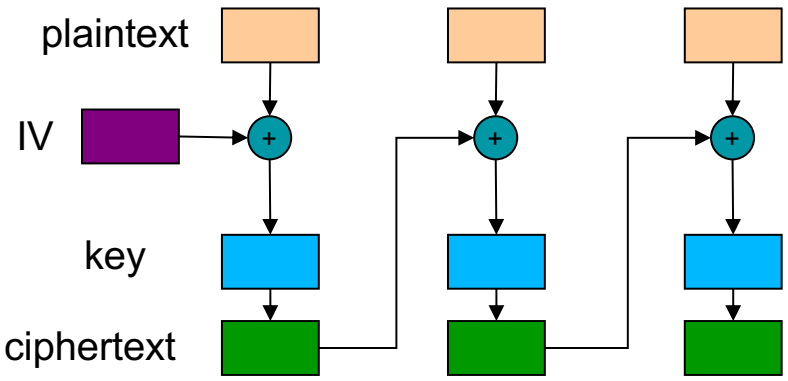
- Mallory can now capture a message transferring money from Alice to Bob and replace parts of that message with her own

3322 12B7 7783 1120 8080 ⇨ 3322 12B7 7783 1120 CDC7

Cipher Block Chaining (CBC)

- Problem with ECB is that Mallory can replace, add or drop blocks at will, and this will not be detected by Alice or Bob
- Chaining prevents this by adding feedback
  - Each ciphertext block depends on all previous blocks
- With CBC, same plaintext blocks will encrypt to different ciphertext blocks thus obscuring patterns in plaintext
- Drawback: Encryption/decryption cannot be parallelized

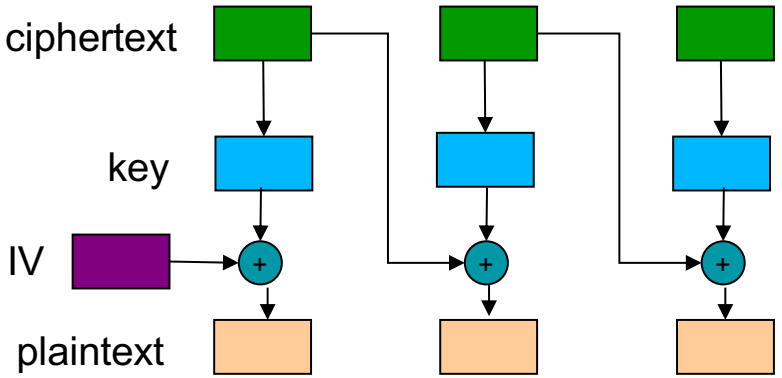
Cipher Block Chaining (CBC)



Encryption

Initialization vector (IV) is just a block of random numbers, to ensure that no messages have the same beginning.
Both the sender and the receiver must use the same IV.
Can be transmitted with the message but must be unpredictable.

Cipher Block Chaining (CBC)



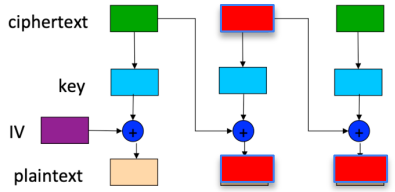
Decryption

CBC and Replay/Modification

Transfer 10$ from X to Y
Transfer 10$ from X to Y ⇨
Transfer 5$ from X to Y

3231 22C1 9928 2F12 0092
8001 1121 7211 AB22 FF21
7272 3B7D 0D98 BBC1 C543

- No structure to learn
  - Same plaintext results in different ciphertext
- Mallory cannot modify parts of the message and not be detected
- Replay is still possible if we don't use timestamps
  - Because IV is sent on the same channel



Try out some algorithms

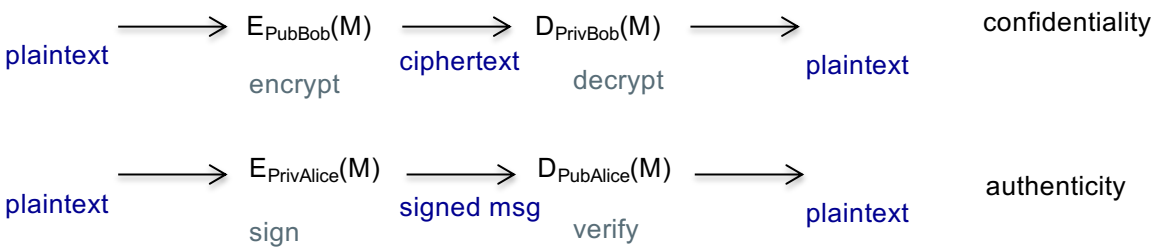
Asymmetric Cryptography

- Everyone has two keys: public and private
- Scramble messages between Alice and Bob so
  - Either: others who see the messages cannot make sense of them (confidentiality) – use Bob's public key
  - Or: everyone can verify who sent the message (authenticity) – use Alice's private key
- Terminology
  - Message to be sent – plaintext
  - Scrambled message – ciphertext or signed message
  - Scrambling process for conf – plain2cipher – encryption
  - Unscrambling process – cipher2plain - decryption
  - Scrambling process for auth – plain2sign – signing
  - Unscrambling process – sign2plain - verification

Asymmetric Cryptography

- Everyone has two keys:
  - Public key K1 that everyone knows
  - Private key K2 that only the owner knows
  - Encryption algorithm and key properties ensure that

$$D_{K2}(E_{K1}(M)) = M$$

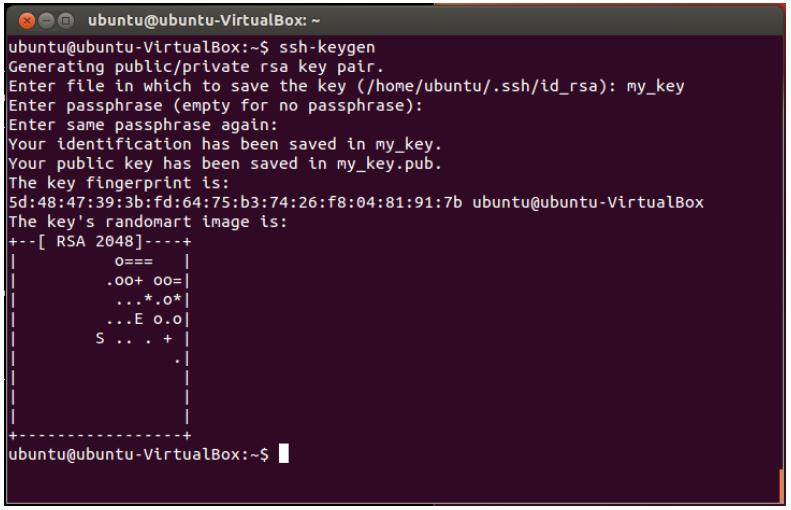


Asymmetric Cryptography

- Requirements for an asymmetric crypto protocol
  1. It is easy to generate a pair of matching keys
  2. It is easy to encrypt/decrypt with each key
  3. It is hard to calculate one key if you know another key
  4. It is hard to break the encryption without knowing the decryption key
  5. Either of the keys in the pair can be used to encrypt/decrypt
- In general functionality of asymmetric crypto (what it can be used for) is greater than that of symmetric crypto
  - But asymmetric operations are ~ 1,500 times slower

Common Asymmetric Algorithms

- RSA (Rivest–Shamir–Adleman)
  - Usage: Encryption, Digital Signatures
  - Key Lengths: Commonly 2048, 3072, or 4096 bits
  - Based on the difficulty of factoring large prime numbers
- DSA (Digital Signature Algorithm)
  - Usage: Digital Signatures
  - Key Lengths: 1024 to 3072 bits
- Diffie-Hellman
  - Usage: Key Exchange
  - Key Lengths: Commonly 2048 bits or higher
  - Allows two parties to each generate a public-private key pair, share their public keys, and then derive a shared secret
- Elliptic Curve Cryptography (ECC)
  - Usage: Encryption, Digital Signatures, Key Exchange
  - Key Lengths: Commonly 256, 384, or 521 bits
  - Provides the same security with shorter key lengths compared to other algorithms



Digital Signatures

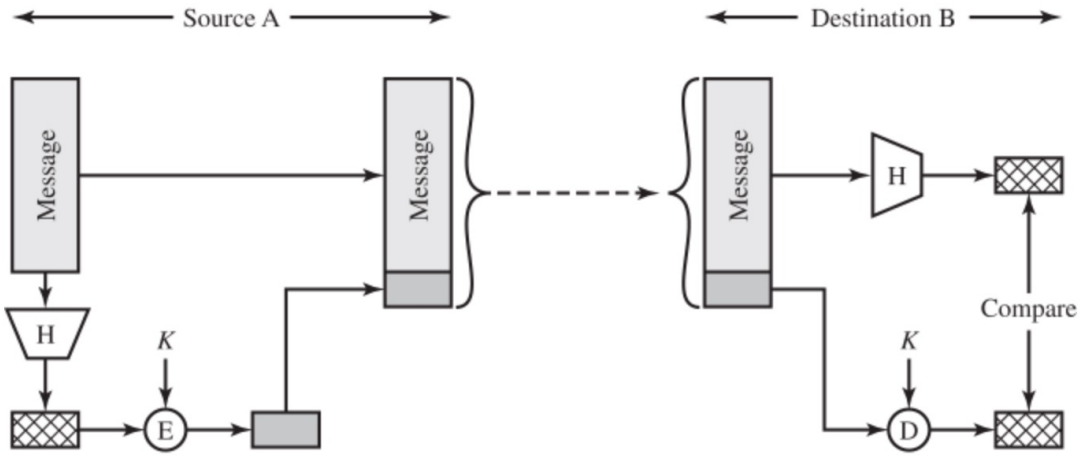
- Signature – encryption of message or its one-way hash with the sender's private key EprivA(H(M))
- Provide non-repudiation
  - Hash the data, encrypt with *private* key
  - Verification uses public key to decrypt hash
  - Calculate the hash of the original message and compare to the decrypted hash
    - Message authenticity (sent by the alleged sender)
    - Message integrity (has not been changed in transit)
- Provide data integrity
  - Can it be done with symmetric systems?
    - Verification requires shared key, cannot provide non-repudiation

Keyed Hashes

- MAC (message authentication code)
  - Appended to a plaintext message to verify its integrity and authenticity
- Hash depends on the message and a key

Keyed Hashes

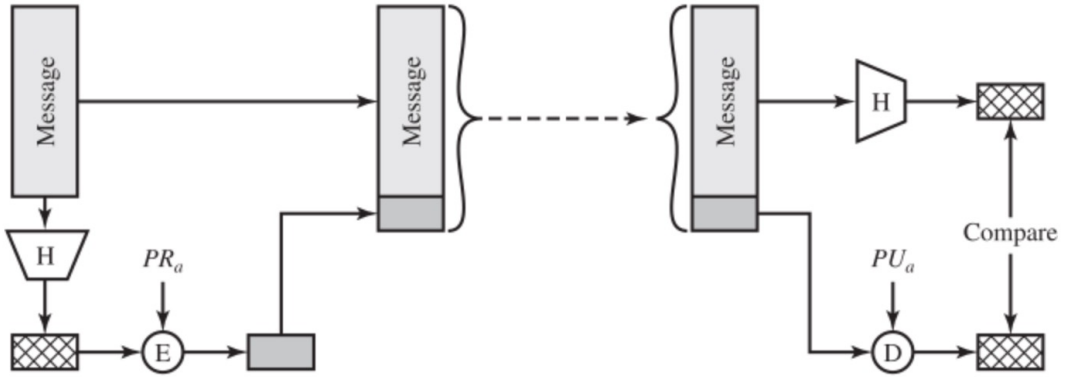
- Using symmetric encryption
  - Integrity and authenticity



(a) Using symmetric encryption

Keyed Hashes

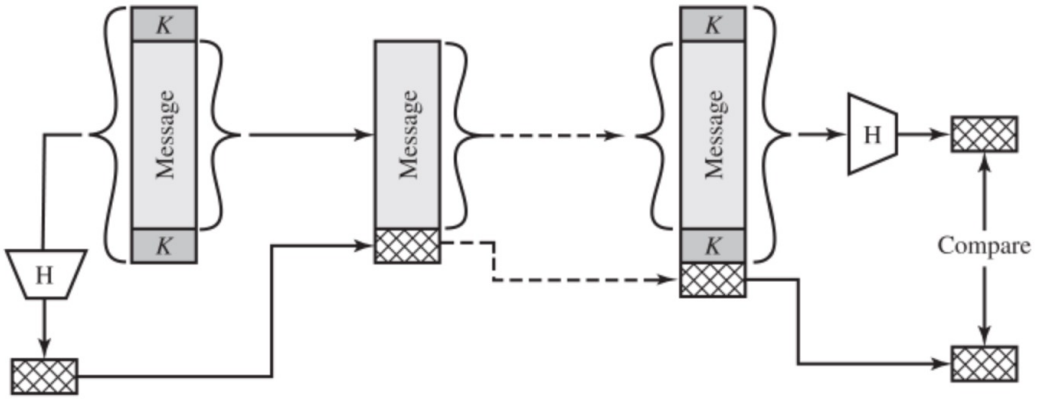
- Using asymmetric encryption
  - Integrity, authenticity, non-repudiation



(b) Using public-key encryption

Keyed Hashes

- Concatenating the shared key
  - Integrity and authenticity



(c) Using secret value

Recap:
Cryptographic Basics Foundational to CPS/IoT Security

- **One-way hash functions:** no key
  - Usually used for **integrity**
  - Plaintext ⇨ fingerprint

e.g., device-to-device data integrity, device password storage, software measurement

- **Symmetric crypto:** one key
  - Usually used for **confidentiality**
  - Both Alice and Bob know the same *shared key*
  - Plaintext ⇨ ciphertext

Faster than asymmetric→ could be useful for resource constraints

- **Asymmetric Crypto:** two keys
  - Usually used for **confidentiality** and **authenticity**
  - Alice has *public key* and *private key*
  - Everyone knows Alice's *public key*, but only Alice knows her *private key*
  - Encrypt with one and decrypt with the other
  - Plaintext ⇨ ciphertext

e.g., digital signatures, secure key exchange, ensure authenticity

Recap:
Cryptographic Basics Foundational to CPS/IoT Security

- **One-way hash functions:** no key
  - Usually used for **integrity**
  - Plaintext ⇨ fingerprint

e.g., device-to-device data integrity, device password storage, software

- **Symmetric**
  - Usually u
  - Both Alic
  - Plaintext

ric→ could be
nstraints

There are many, many, many more crypto topics that we'll cover on an as-needed basis…

- **Asymmetri**
  - Usually u
  - Alice has
  - Everyone
  - Encrypt with one and decrypt with the other
  - Plaintext ⇨ ciphertext

e.g., digital signatures, secure key exchange, ensure authenticity


Questions?