

EE 4215

Security Technology

Lecture 11: Software Security & Revision
2018/19 Semester B

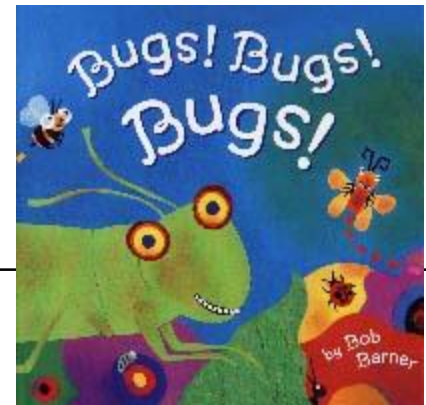
Course Structure

- Week 1: Security Fundamental
- Week 2: Classical Cryptography
- Week 3: Number Theory &
- Week 4: DES
- Week 5: Secret-key Cryptography
- Week 6: Public-key Cryptography
- Week 7: Quiz 1
- Week 8: Hashing, Digital Signature, PKI
- Week 9: SSL/TLS, Web Security
- Week 10: Wi-Fi & P2P Security, Blockchain Security
- Week 11: Smart-card Security
- Week 12: Cloud Computing Security
- Week 13: Quiz 2
- Week 13: Secure-code, Trusted Hardware, Review

Software Bugs

- Possible consequences of buggy software
 - Threat to life
 - Therac-25 (1985-1987) –radiation therapy machine killed 3 patients
 - Threat to the society
 - Power outage occurred throughout parts of the Northeastern and Midwestern United States and the Canadian province of Ontario on Thursday, August 14, 2003 was caused by a software bug in the alarm system at a control room.
- Possible financial loss to the organization
 - In 2011, more than 40 Commonwealth Bank cash machines in Sydney were affected by a problem and could not identify the account balances of customers. Customers who had little or no money were able to withdraw large sums of cash.

Bug density?

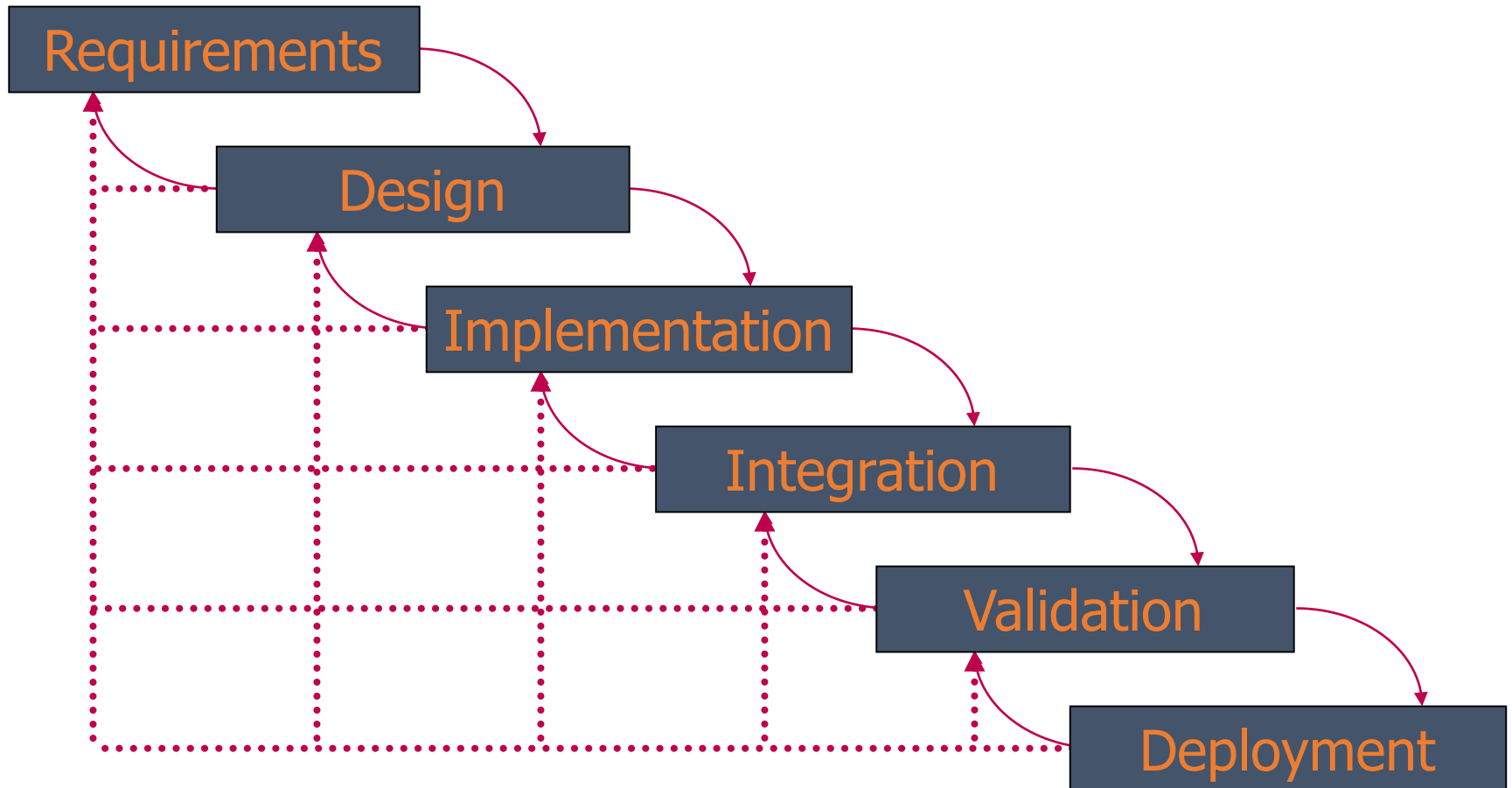


- 10 bugs per 1,000 lines of code?
 - Million lines of code → 10,000 bugs
- Complexity breeds bugs

Lines of code (millions)	
NT 3.1	6
NT3.5	10
NT 4.	16
Windows 2000	29
Windows XP	40
Windows Vista	50
RedHat 7.1	30
Linux 6.0 kernel	6
Debian 2.2	56
Debian 3.1	213
Mac OS X 10.4	86
OpenSSL	0.2
Apache	0.2

Software Development Lifecycle

Waterfall Model

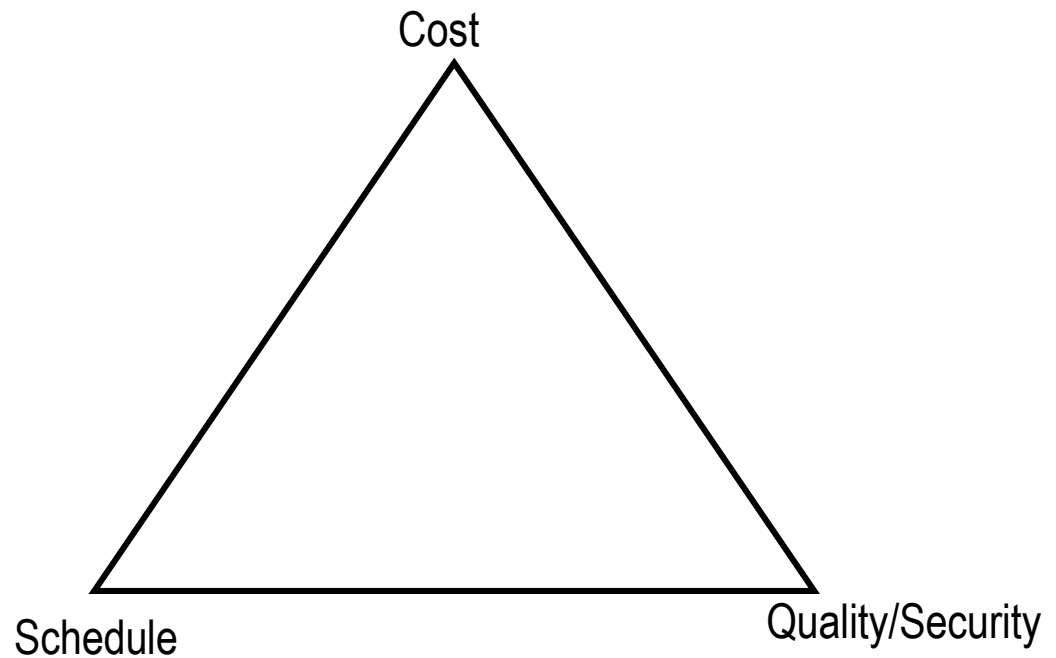


Why is software development so hard?

- Complex
 - No two parts the same
 - Growth adds to complexity in non-linear fashion
 - Interaction between data, function, user, devices
- Changeable
 - Easy to change, so let's change it
- Human intensive
- Market pressures (time to market)
 - Eliminate all bugs, never ship
 - Ship now, let customers debug

Dilemma

- Security conflicts with
 - Ease of use
 - Speed
 - Feature richness



Pick any two.

Software Development

▶ Individual / small team

- Single developer / small team
- “Toy” applications
- Short lifespan
- Single or few stakeholders
 - Architect = Developer = Manager = Tester = Customer = User
- Built from scratch
- Minimal maintenance

▶ Enterprise strength

- Teams of developers with multiple roles
- Complex systems
- Indefinite lifespan
- Numerous stakeholders
 - Architect ≠ Developer ≠ Manager ≠ Tester ≠ Customer ≠ User
- Reuse to minimize costs
- Maintenance accounts for over 60% of overall development costs

Playing the software development game ...



- The trusting model
 - Design/code/test to provide proper functions
 - Bugs are random and rare
- The defensive model
 - Design/code knowing that you have an active adversary
 - Test not only that the right things happen, but also that wrong things don't happen
 - Bugs may be vulnerabilities and exploited



Insecure Security

- Attackers might deliberately abuse software
 - To gain unauthorised access
 - To gain financial advantages
 - E.g. how to hack an ATM ()
 - ...

How to Make an ATM Spit Out Money

A computer security researcher demonstrates attacks on cash machines.

Rectangular Snip

By Erica Naone on July 29, 2010



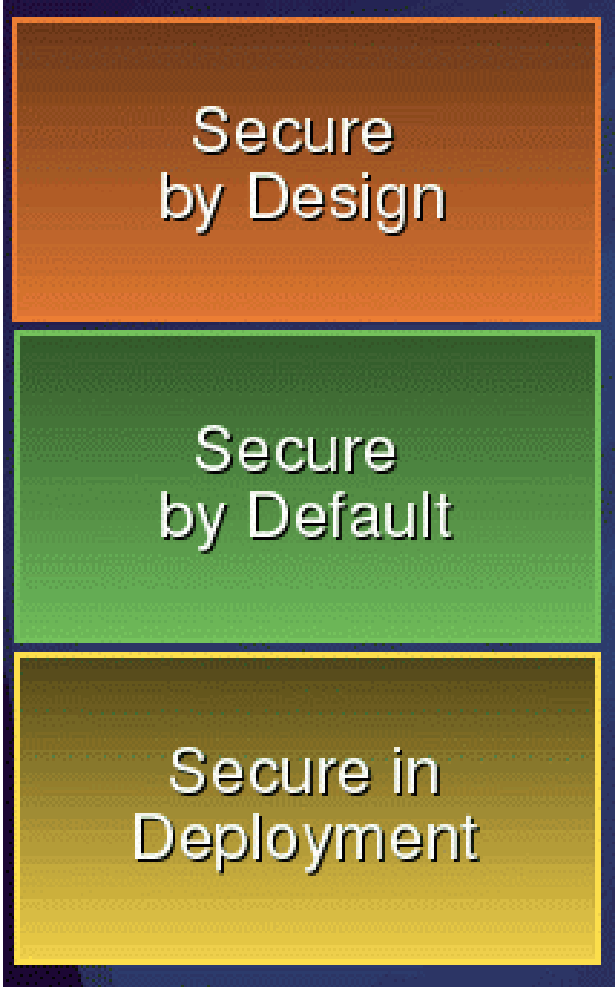
Yesterday, during a flashy presentation at the Black Hat security conference in Las Vegas, a computer security expert showed several ways to break into ATMs.

Barnaby Jack, who is director of research at [IOActive Labs](#), made cash pour from a machine for minutes on end. After studying four different companies' models, he said, "every ATM I've looked at, I've found a 'game over' vulnerability that allowed me to get cash from the machine." He's even identified an Internet-based attack that requires no physical access.



Software engineering for security

- Secure design
- Secure implementation
- Security testing
- Secure deployment
- Root cause analysis for bugs
- Policy and training



Secure
by Design

Secure
by Default

Secure
in
Deployment

Secure design

- Assess the risk
 - Detail your assets
 - Know the threats and your attackers
 - Costs: time & money ... acceptable risk
- Formal threat modeling
 - Attack trees
 - Reduce the attack surface
- Apply secure design principles
- Think like a bad guy

How are the bad guys finding bugs?

- Open source → code review
 - grep for default secrets, insecure APIs
 - Analyze input and packet parsing
- Blackbox testing
 - “random” alteration of input – big username, missing fields ...
 - Cause a crash
 - Analyze dump
 - experiment
 - Disassembly
 - Look for `str*()`, network code
- Bug announcements, or diff patched vs old images
 - Attack unpatched versions



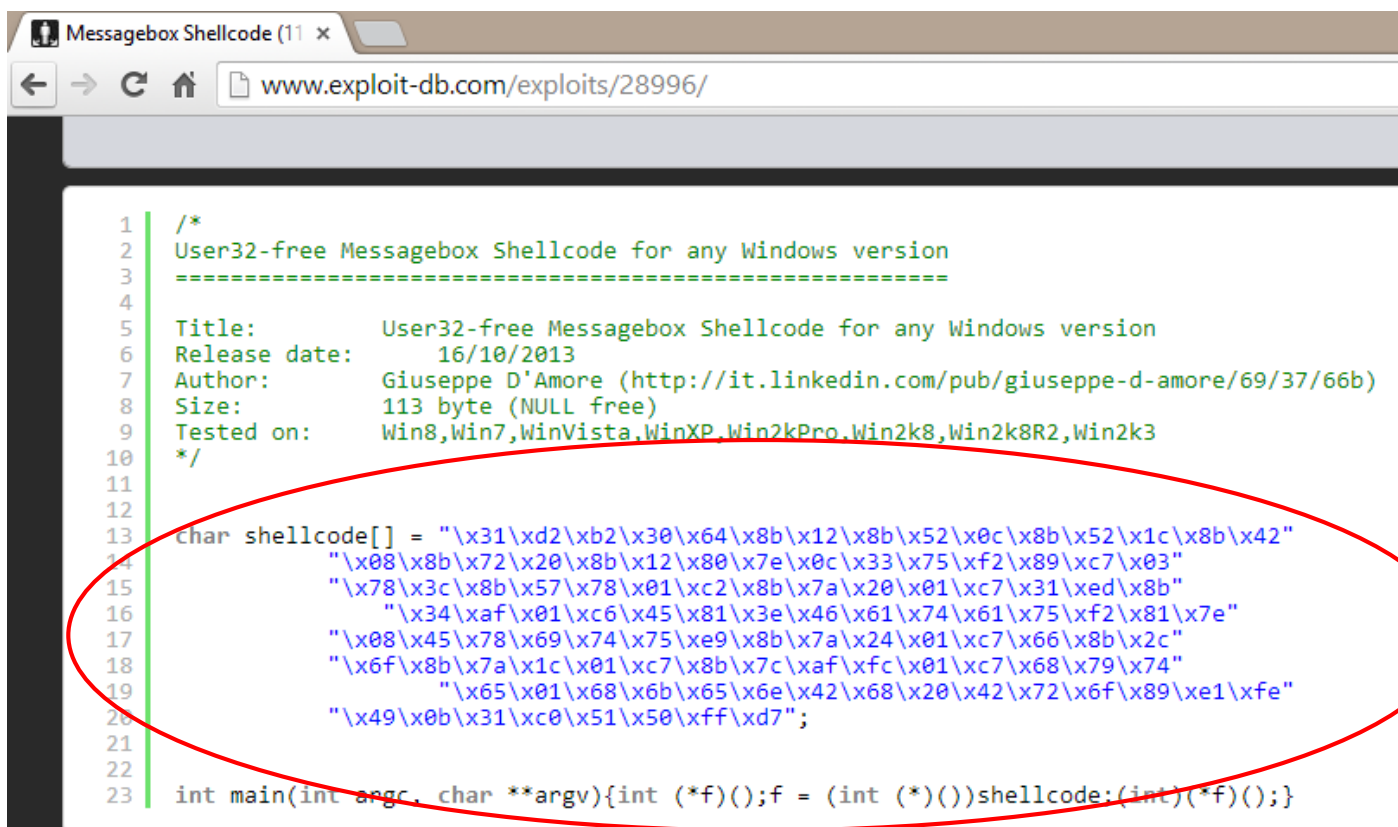
Common Methods

- Buffer Overflow

- Escalating a normal user to root / administrative privilege by exploiting a buffer overflow that exists in a vulnerable program, in particular, C program
 - Buffer is the memory location where data is stored
 - Overflow: data exceeded the maximum storage allocated to the variable
- The exploitable program must be run in a privileged user
 - Therefore the best practice is not to run all programs using root / administrator
- strcpy(), printf(), etc C functions are vulnerable

Buffer Overflow

- The exploit was done by a short piece of code called “shell code” aiming to starts command shell from which the attacker can control the compromised machine



```
1  /*
2  User32-free MessageBox Shellcode for any Windows version
3  =====
4
5  Title:      User32-free MessageBox Shellcode for any Windows version
6  Release date:  16/10/2013
7  Author:      Giuseppe D'Amore (http://it.linkedin.com/pub/giuseppe-d-amore/69/37/66b)
8  Size:        113 byte (NULL free)
9  Tested on:    Win8,Win7,WinVista,WinXP,Win2kPro,Win2k8,Win2k8R2,Win2k3
10 */
11
12
13 char shellcode[] = "\x31\xd2\xb2\x30\x64\x8b\x12\x8b\x52\x0c\x8b\x52\x1c\x8b\x42"
14                    "\x08\x8b\x72\x20\x8b\x12\x80\x7e\x0c\x33\x75\xf2\x89\xc7\x03"
15                    "\x78\x3c\x8b\x57\x78\x01\xc2\x8b\x7a\x20\x01\xc7\x31\xed\x8b"
16                    "\x34\xaf\x01\xc6\x45\x81\x3e\x46\x61\x74\x61\x75\xf2\x81\x7e"
17                    "\x08\x45\x78\x69\x74\x75\xe9\x8b\x7a\x24\x01\xc7\x66\x8b\x2c"
18                    "\x6f\x8b\x7a\x1c\x01\xc7\x8b\x7c\xaf\xfc\x01\xc7\x68\x79\x74"
19                    "\x65\x01\x68\x6b\x65\x6e\x42\x68\x20\x42\x72\x6f\x89\xe1\xfe"
20                    "\x49\x0b\x31\xc0\x51\x50\xff\xd7";
21
22
23 int main(int argc, char **argv){int (*f)();f = (int (*)(void))shellcode;(int)(f)();}
```

Common Methods

- Decompiling the programs
 - This is a kind of static analysis
 - Java code can be decompiled
 - Android apk which is basically Java can also be decompiled
 - Native exe programs would be hard for decompilation, in particular when the symbol table is stripped away
 - Products are available to assist the reverse-engineering work

Recommendations

- Pick a secure programming language
 - E.g. one which is less vulnerable to buffer overflow
- Never store plain-text encryption keys / password in program source code
- Never store keys in specifically named resources
 - Attacker might be interested to look at file named *serverkey.txt* or a key stored in a plist with a name which contains key or password / passwd

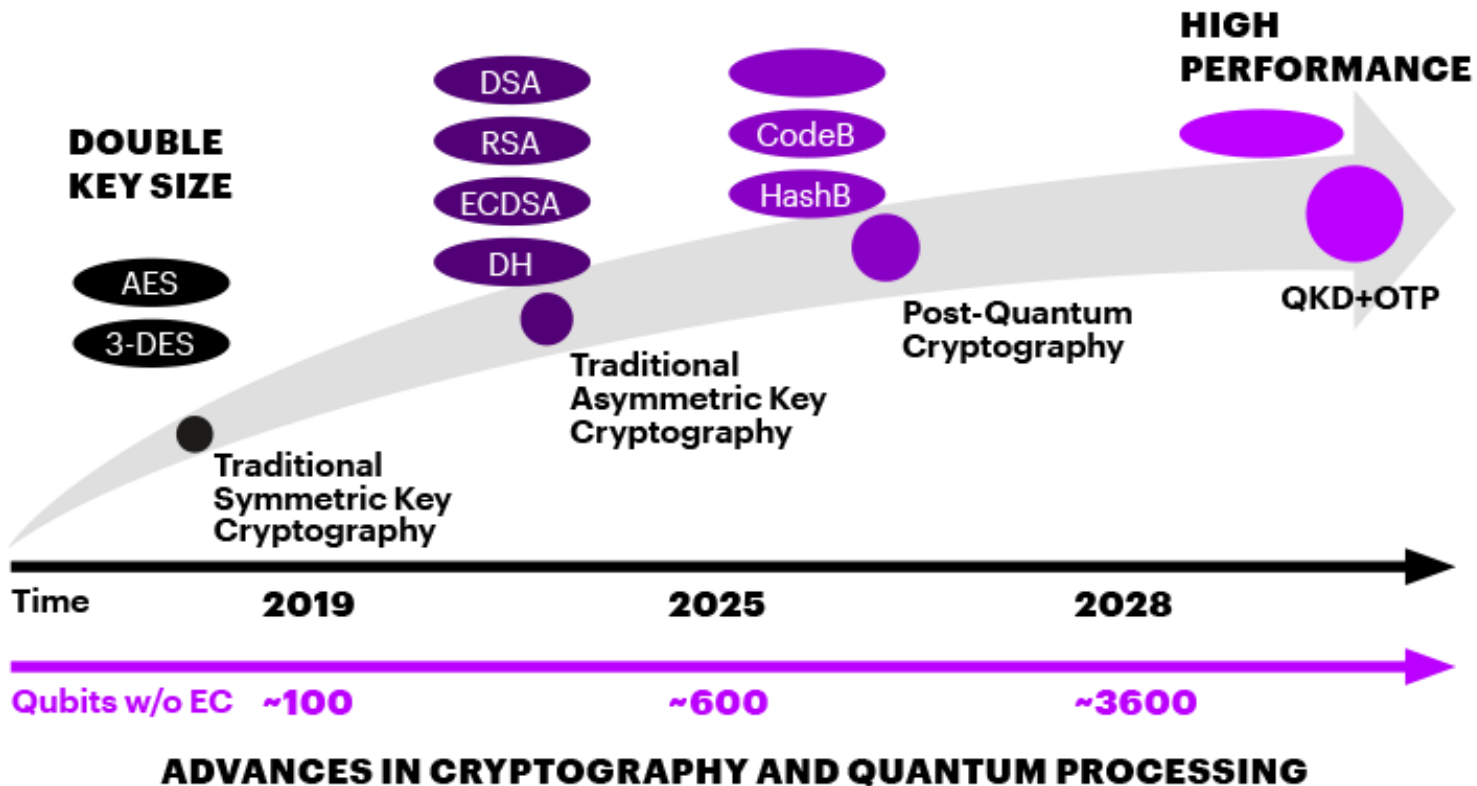
Advanced Topics

- Post-Quantum Cryptography
- Blockchain Security
- Hardware Security
- Trusted Hardware

Post-Quantum Cryptography

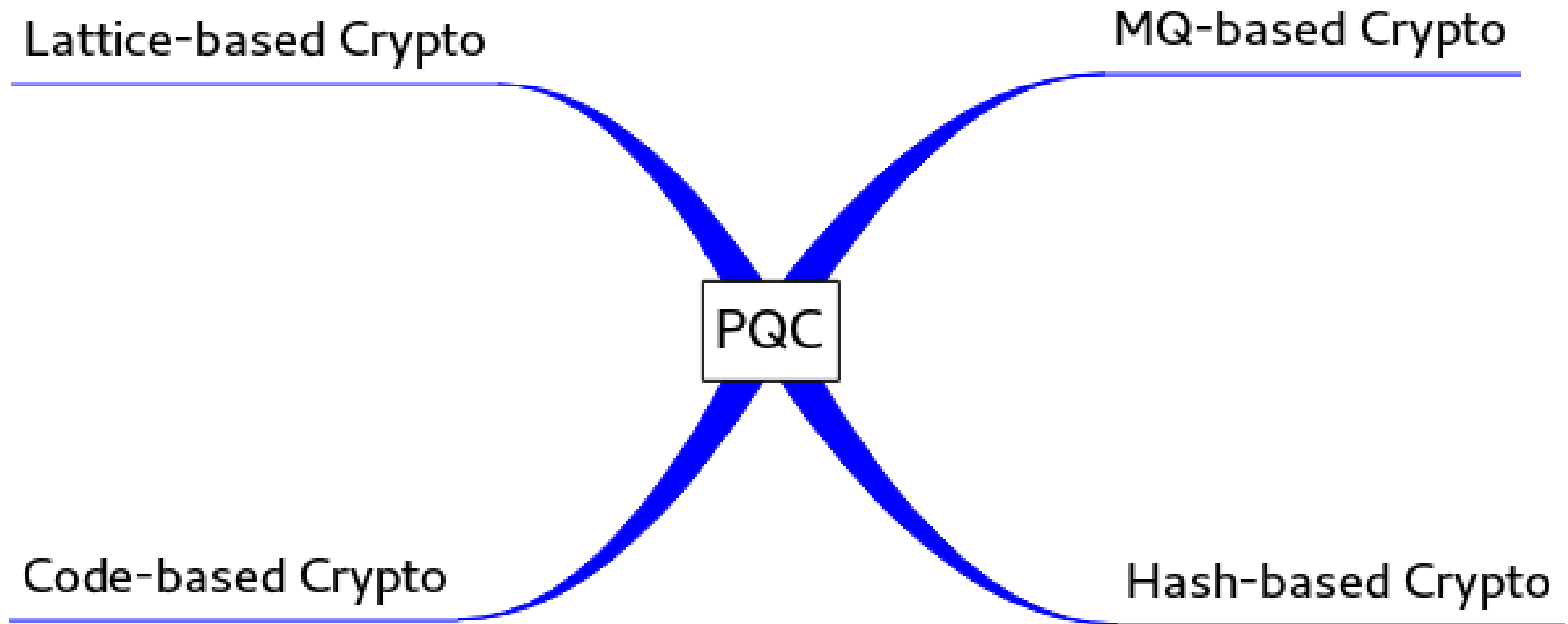
- Source: Accenture
- Quantum computers → 10 to 15 years from now?

FIGURE 4. Timeline for future standardization events (copyright Accenture)



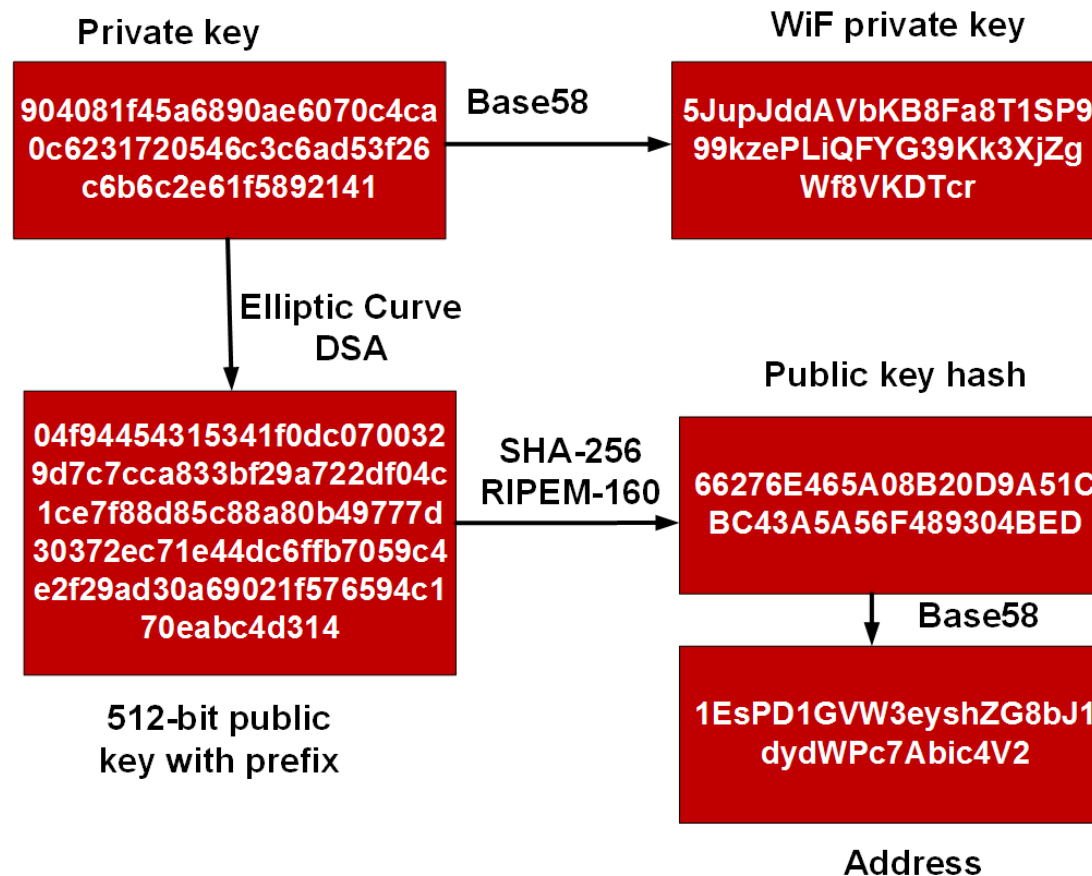
PQC

- Lattice-based is more common.



Blockchain Security

- Blockchain uses ECC to sign transactions.
- WiF = Wallet Interchange Format, Bitcoin ID as follow



Example

- You can generate your own Bitcoin ID:

Private key:

660be9373586e898e70c5534d801534d06e91ae1bfb3886c6f12ed87a37dd2b9

Public key:

040a6a2d6f14870ed589b5a015c67a28612733e3de57911c05f6d2fc5caee71d7fb
bc56b34150f3b86ec91f65e28cef7f032fc52f7e35301e6a31cf50a081e019a

wif: 5JbEDtj2tKFLbfgwqfMVeYJYqgAp39Vjs6nJDeqX8rC6nYESzSo

Address: 13EiXzu7nFvar2ejbpbcBNi3LRpda3xFVt

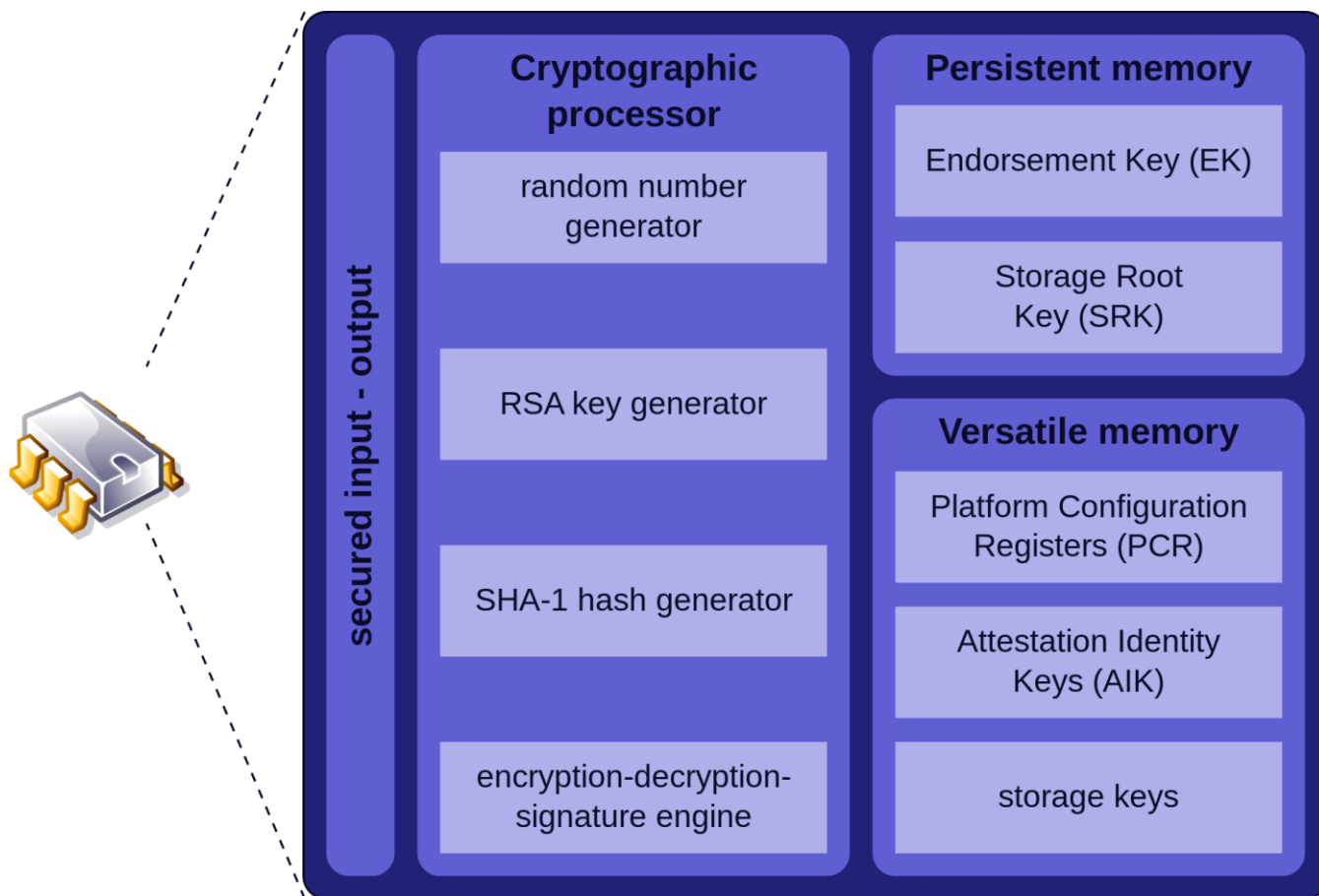
Signed:

4830450221008c37d448ccf17aef84fba79630fd67495e92f71c8aedb16906a2fb2
fb90619d602205ea6e4584967aed884a82494aa2e25195e19108351190cb6d940d3
14b38480f301

41040a6a2d6f14870ed589b5a015c67a28612733e3de57911c05f6d2fc5caee71d7
fbbc56b34150f3b86ec91f65e28cef7f032fc52f7e35301e6a31cf50a081e019a

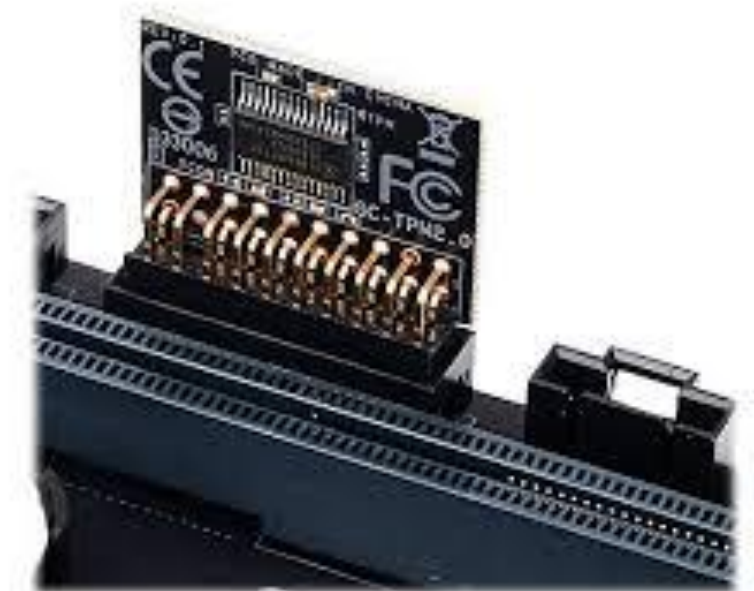
Trusted Platform Module (TPM)

a dedicated microcontroller designed to secure hardware through integrated cryptographic keys

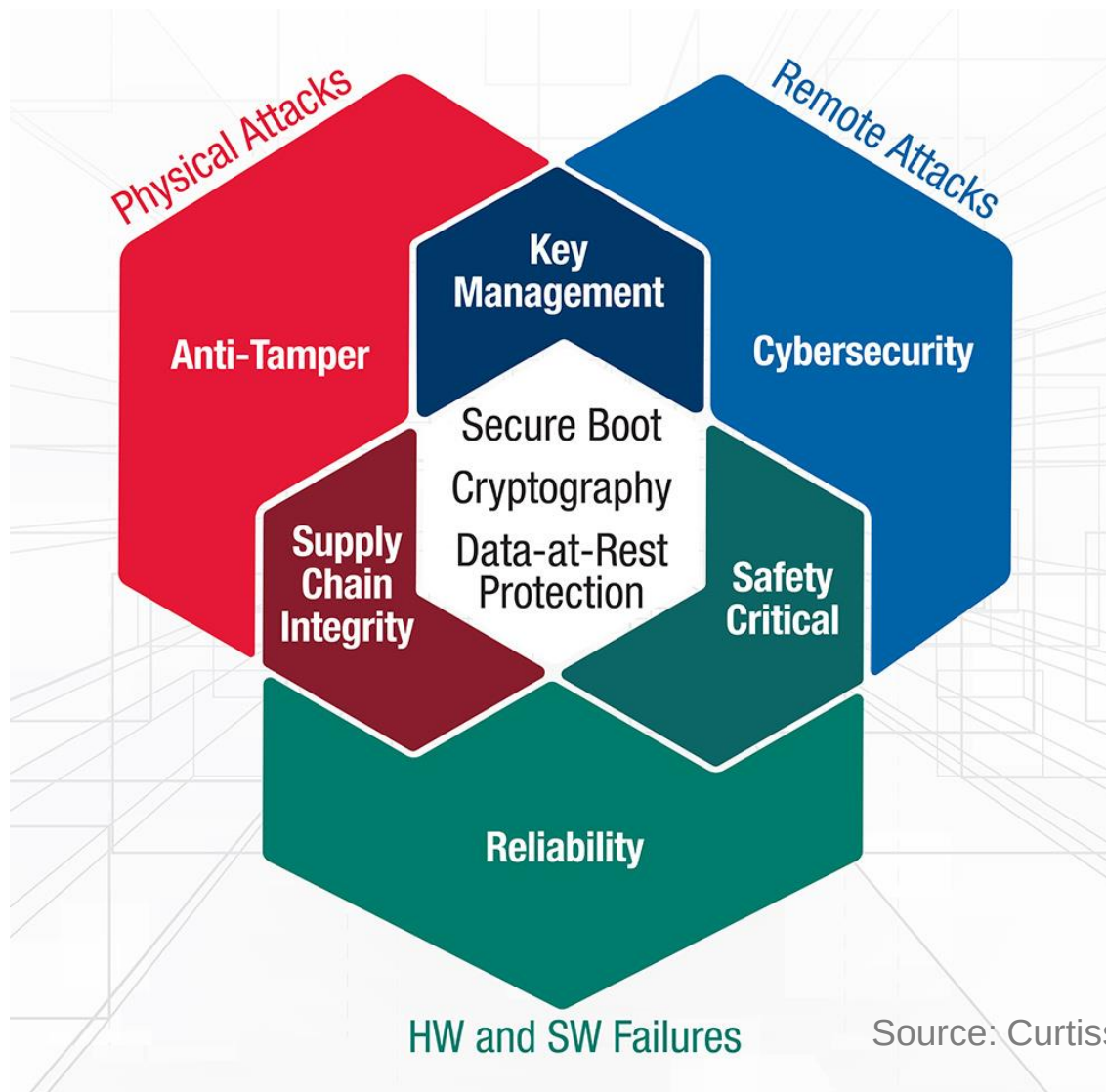


Existing Products

- Around \$20 USD:
<https://www.amazon.com/slp/tpm-chip/2rhwms3q26ja4tz>



Trusted Computing



- Endorsement key
- Secure input and output
- Memory curtaining / protected execution
- Sealed storage
- Remote attestation
- Trusted Third Party
- Loss of anonymity

Course Review

What we have covered

- You are expected to know & understand the basic terms such as confidentiality, integrity, authentication, etc
- The materials on cryptography form the basis for other topics in this course
 - Classical ciphers (substitution cipher & transposition cipher) and one-time pad
 - Symmetric encryption / asymmetric encryption algorithms
 - Efficient implementation
 - Attack approaches
 - Hashing
 - Digital signature, digital certificate and SSL / IPsec

What we have covered

- Be sure to know how to use *openssl*, *WireShark*
- Understand the components used in enterprise infrastructure and the concept of defense in depth
- Understand how web works and the threats to web application
- Wireless network and its protection
- Network sniffing
- Smartcard technology
- Cloud technology



Good Luck to your Final Exam!
60% of your Final Grade!
Short Question + Long Questions

