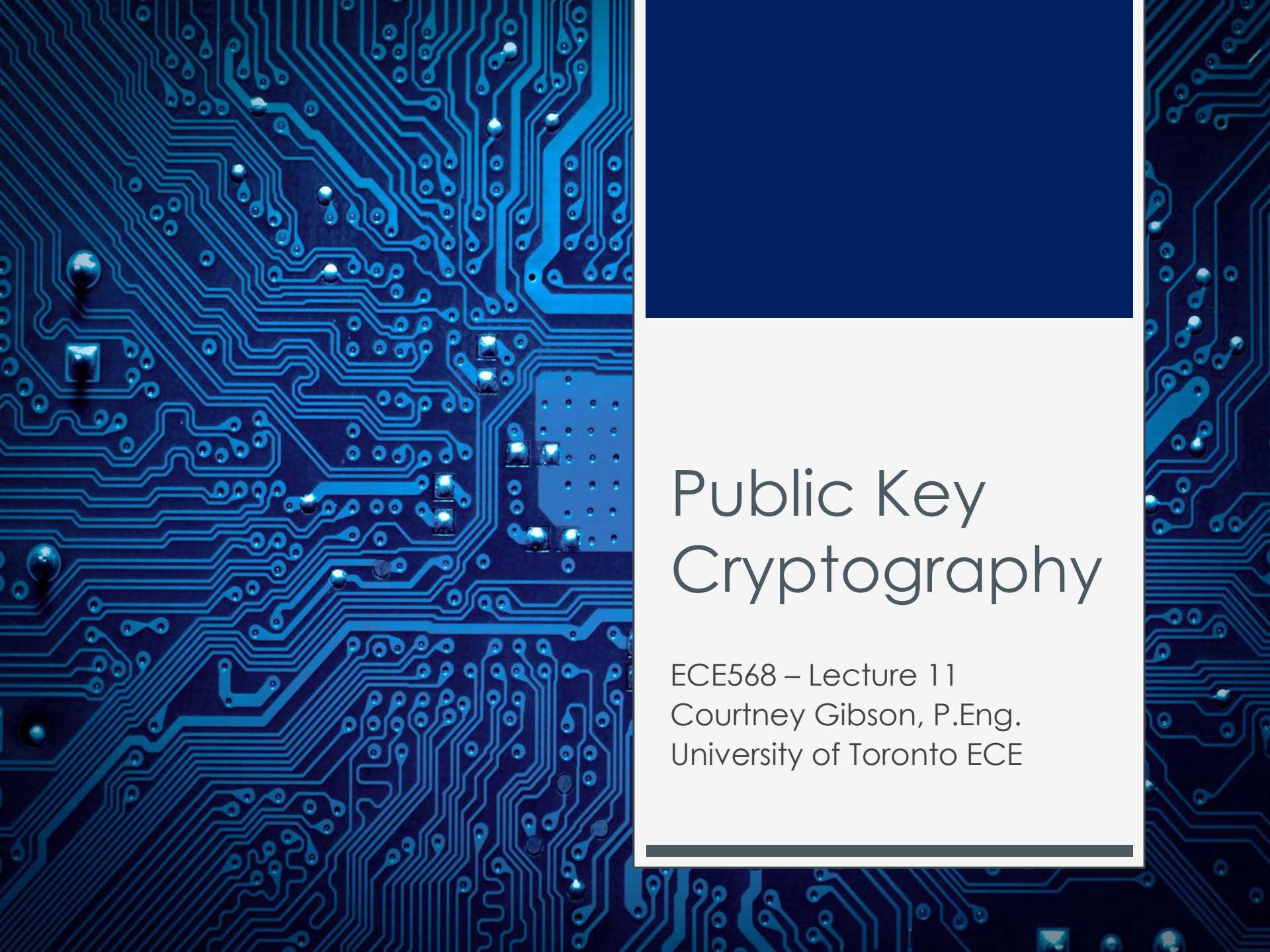


Public Key Cryptography

ECE568 – Lecture 11

Courtney Gibson, P.Eng.
University of Toronto ECE

Outline

Introduction

- Encryption/Decryption
- Authentication

RSA Algorithm

- Background
- Operation

Public-Key Infrastructure (PKI)

- Digital Signatures and Certificates
- PKI and PGP
- Certificate Revocation

Introduction

Encryption, decryption, authentication

Public Key Cryptosystems

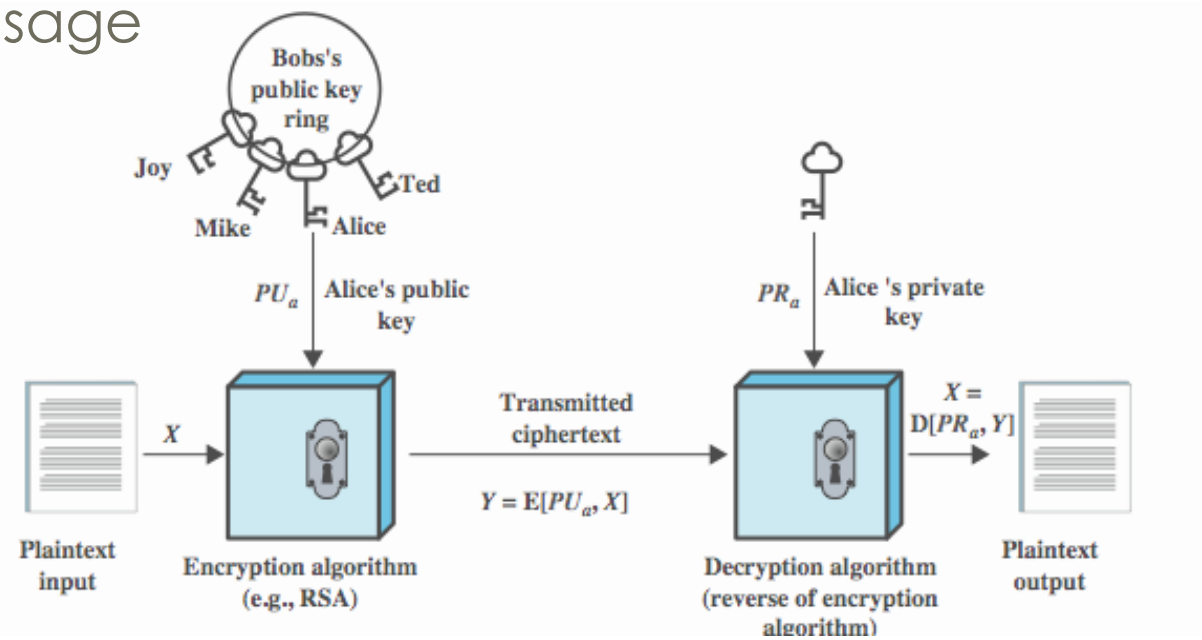
Public Key cryptosystems use a **pair** of keys:

- Every user has a public/private key pair
- The private and public key reveal nothing about each other
- Users distribute the public key, while keeping the private key in a safe place
- Messages encrypted with one key can only be decrypted with the other key

Public Key Encryption

Encryption: the sender encrypts the message with the intended recipient's public key

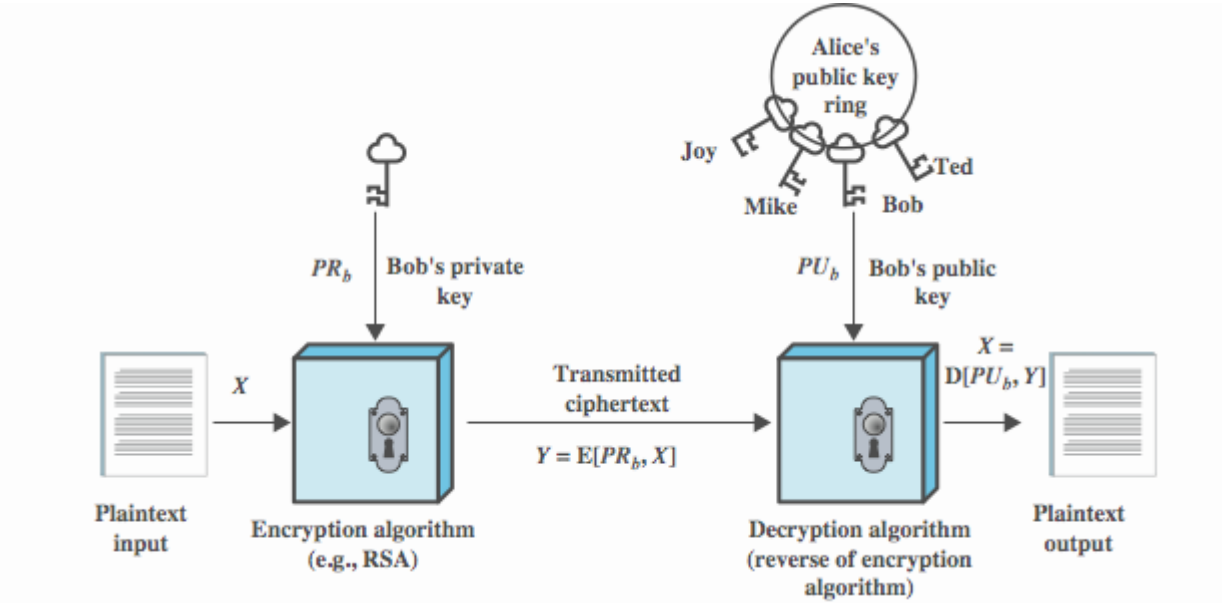
- Only the recipient should have the private key, so only the recipient can decrypt the message



Public Key Authentication

- For authentication, the message is encrypted with the sender's private key (also called **signing**)
  - Any recipient can decrypt using sender's public key
  - Only sender could have encrypted the text we received, thus providing **authentication** and **non-repudiation**
  - Example application: e-mail

Public Key Authentication



RSA

Algorithm, Extended Euclidian Method, limits of RSA

The RSA Algorithm

- RSA is a popular public key algorithm
- RSA was first published by Ron Rivest, Adi Shamir and Len Adleman at MIT in 1977
- They subsequently founded RSA Corporation, which licenses cryptographic protocols and provides security products
- RSA was patented in the US, but the patent expired in 2000
- There is evidence that the GCHQ in London (British equivalent of the American NSA) had invented a similar algorithm as early as 1973

The RSA Algorithm

Step 1: Pick our Primes

RSA is based on modular arithmetic. We need a value, **n**, that we can use as the basis of our modular space:

- RSA key generation starts by randomly picking two *very* large prime numbers: **p** and **q**.
- Define **n** = (**p•q**)

Observation #1: The size of **n** defines the key size: 4096-bit RSA uses 4096 bits to represent **n**.

Observation #2: we can publicly share the value **n**, because there is no known algorithm to efficiently factor it and recover **p** and **q**.

The RSA Algorithm

Step 2: Pick our Public Key

Euler's Theorem (we'll come back to this) makes it convenient to base our public and private keys off a value φ:

- Define $\varphi = (\mathbf{p}-1)(\mathbf{q}-1)$
- Pick a random value, **e**, that is *coprime*[1] to φ. **e** will be our ***public key***.

[1]Two numbers, **a** and **b**, are *coprime* (or *mutually prime*), if 1 is the only positive integer that evenly divides both of them.

The RSA Algorithm

Step 3: Encrypt our Message

Someone else can use our public key (**e**) to encrypt a *plaintext message*, **M**, into a *cryptotext message*, **C**:

$$\mathbf{C} = \mathbf{M}^{\mathbf{e}} \bmod \mathbf{n}$$

The values, **e** (*public key*), **n** (*modulus*) and **C** (*cryptotext message*) can be safely made public. The values **p**, **q** and φ must remain secret.

Observation: M < n (or else RSA does not work).

The RSA Algorithm

Step 4: Calculate our Private Key

If we want to decrypt the cryptotext message that was sent to us, then we need a **private key** that can "undo" our **public key**.

Specifically, we need to calculate a **private key** (**d**) that is the *multiplicative inverse* of our **public key** (**e**):

$$(\mathbf{e} \bullet \mathbf{d}) = 1 \bmod \varphi$$

This value for **d** can be efficiently computed using the *Extended Euclidean Method*[1]. (And, as the name suggests, **d** must remain private.) Nobody else can calculate **d**, because they do not know φ.

[1]You are not required to memorize this algorithm; for more detail on how it works, see the *Handbook of Applied Cryptography* or Wikipedia.

The RSA Algorithm

Step 5: Decrypt with our Private Key

We recover the original message, **M**, using our *private key* (**d**):

$$\begin{aligned} \mathbf{M} &= \mathbf{C}^{\mathbf{d}} \bmod \mathbf{n} \\ &= (\mathbf{M}^{\mathbf{e}} \bmod \mathbf{n})^{\mathbf{d}} \bmod \mathbf{n} \\ &= \mathbf{M}^{\mathbf{e}\cdot\mathbf{d}} \bmod \mathbf{n} \end{aligned}$$

Recall that $(\mathbf{e}\cdot\mathbf{d} = 1) \bmod \varphi$. Therefore, $(\mathbf{e}\cdot\mathbf{d}) = (k\varphi + 1)$ for some integer k.

$$\begin{aligned} &= \mathbf{M}^{k\varphi+1} \bmod \mathbf{n} \\ &= \mathbf{M}^{k\varphi} \cdot \mathbf{M} \bmod \mathbf{n} \\ &= (\mathbf{M}^{k\varphi} \bmod \mathbf{n}) \cdot (\mathbf{M} \bmod \mathbf{n}) \end{aligned}$$

Euler's Theorem tells us that $a^{\varphi} = 1 \bmod \mathbf{n}$.

$$\begin{aligned} &= 1 \cdot (\mathbf{M} \bmod \mathbf{n}) \\ &= \mathbf{M} \text{ (assuming } \mathbf{M} < \mathbf{n}) \end{aligned}$$

Improper use of RSA

- Recall that a message is **signed** by encrypting the message with the sender's private key
- RSA has very poor resistance to spoofing because encryption uses exponentiation

$$\begin{aligned} \text{encrypt}(K \bullet M) &= (K \bullet M)^d \quad \text{// d is private key} \\ &= K^d \bullet M^d \\ &= \text{encrypt}(K) \bullet \text{encrypt}(M) \end{aligned}$$

If someone will sign messages the adversary gives them, then she can trick them into signing messages they have never seen:

- Suppose a victim will not sign message **M**, but the adversary can pick a **K** and get the victim to sign **K•M** and **K**, then a signature on **M** can be recovered

Public-Key Infrastructure (PKI)

Key-signing authorities, key revocation, PGP "web of trust"

Public Key Infrastructure

Does public key cryptography prevent man-in-the-middle attacks?

- If Alice wants to share a key with Bob, encrypting it with Bob's public key prevents Mallory from getting the key
- What if Mallory arranges for Alice to get Mallory's public key, but makes her think that it's Bob's key?
  - Then Alice will encrypt whatever message she wants to send to Bob with Mallory's key
  - Bob won't be able to decrypt and might complain to Alice
  - However, damage is already done, since Mallory can decrypt Alice's message to Bob
- Public Key Infrastructure (PKI) solves this problem

Public Key Infrastructure

PKI is a system where a **trusted third party** (a principal that everyone trusts) vouches for the identity of a key (*i.e.*, that the key belongs to a principle)

- Example:
  - **Assume** that Alice and Bob trust Trent
  - **Assume** that everyone knows Trent's public key
  - Bob creates a public key and goes to Trent; Trent sees both Bob and his public key, creates a **certificate** that says "This public key xxx belongs to Bob" and **signs** it with his (Trent's) private key
  - Bob sends Alice his own public key along with the certificate that bears Trent's signature
  - Alice uses Trent's public key and the certificate to verify Bob's public key

Public Key Infrastructure

- Mallory cannot pretend her key is Bob's key
  - Mallory cannot ask Trent to give her a certificate claiming her key is Bob's key (Trent will only give her a certificate that says the key belongs to Mallory)
  - Mallory cannot forge (or fake) Trent's signature and thus cannot fake a certificate that says her key belongs to Bob

Public Key Infrastructure

Common standard format for certificates is X509

- This format is used in SSL (Lab 3)
- PKI allows using a **chain of certificates** issued by a hierarchy of CAs

Are we back to the **trusted central server** for key exchange? What's the difference in this case?

- Trust level
- Availability, integrity

Certificate Authorities

In the real world, a **Certificate Authority** (CA) plays the role of "Trent"

- Several major CAs (Verisign, Entrust, Equifax, etc.)
- PKI is used within many large organizations

When a browser connects to a secure website, the website sends the browser a certificate that you can verify by viewing the certificate.

PGP: An Alternative to PKI

Instead of having a central trusted party, Pretty Good Privacy (PGP) uses a **web of trust**:

- Every user has a public/private key pair and is capable of signing certificates
- If user Alice is able to verify that a certain public key really belongs to Bob, she can sign a certificate saying so with her private key
- Similarly, if Charlie can verify Alice's public key then he can sign it with his private key
- Trust is transitive: if you trust Charlie, then you can trust Alice, and Bob. If you only trust Alice, then you trust Bob, but not Charlie.

Certificate Revocation

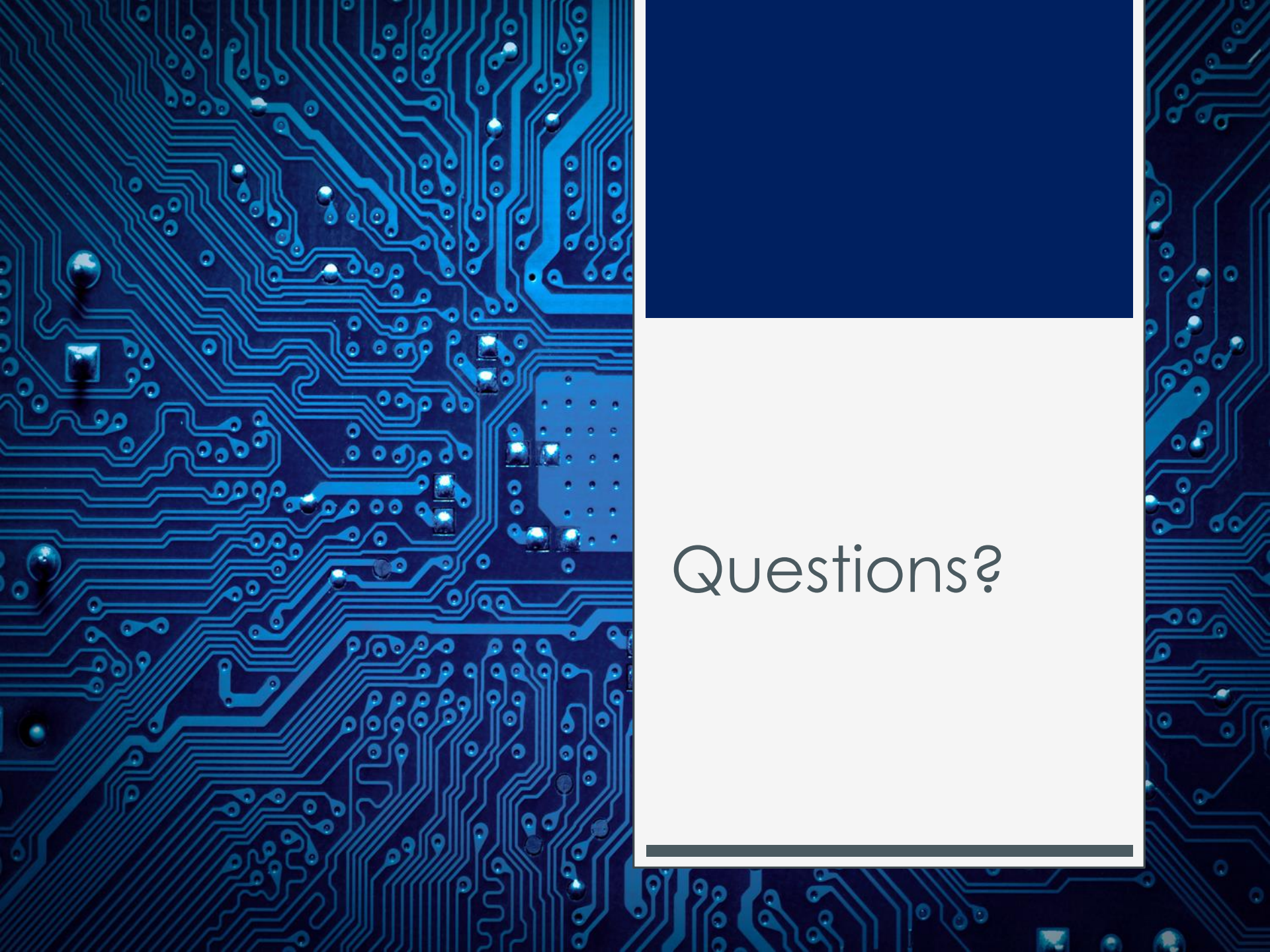
An important aspect of an certificate scheme is the ability to **revoke** certificates:

- Say Microsoft gets a public key certificate from Verisign
- Some hacker is able to steal Microsoft's private key
- The hacker can now create software releases, signed with Microsoft's private key, and the software will appear authentic to the end users' systems
- Microsoft must tell everyone to stop using Microsoft's Public Key to verify signed Microsoft products

- Microsoft uses a **revocation certificate**
  - Certificate should be signed by Verisign (why?)

The revocation certificate is usually created at the time the public key is signed by Verisign

- Certificate should be stored safely (why?)


Questions?