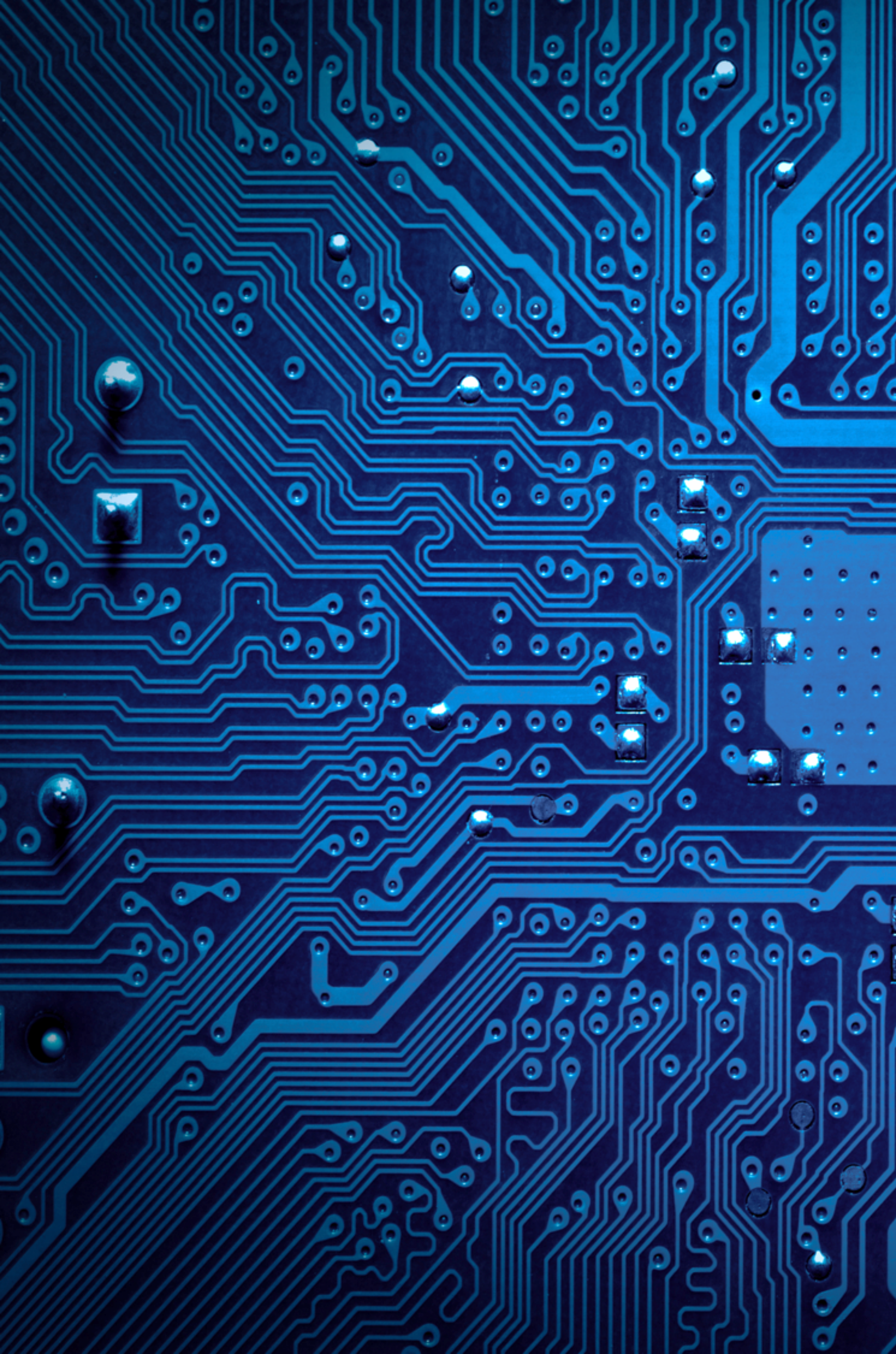


Key Exchange

ECE568 – Lecture 10

Courtney Gibson, P.Eng.
University of Toronto ECE

Outline

Trusted Server
- Needham-Schroeder
- Kerberos

Diffie-Hellman Key Exchange
- Finite Fields
- Modular arithmetic
- Attacks on Diffie-Hellman

Public-Key Based Key Exchange
- Introduction

Pre-Shared Keys

- The **symmetric ciphers** we've looked at thus far allow confidential communication between two parties that share a secret key
- However, the key must be communicated securely between the two parties, over a trusted channel, before decryption can begin
- If an adversary is able to intercept the transmission of the key, the security of the communication is lost

Motivation for Key Exchange

If you want the ability for any **two** parties to conduct private communication, then the costs of exchanging pre-shared keys grows quadratically with the population:

- A population of **N** people needs a total of **N(N-1) ÷ 2** keys
  - 10 people = 45 keys
  - 100 people = 4950 keys
- Infeasible for any large-scale community
  - High cost
  - Many opportunities for interception

Key Exchange

- **Key exchange** deals with establishing a shared secret across an **insecure** channel
- There are three common methods for key exchange that we will examine:
  1. Trusted third-party
  2. Diffie-Hellman Key Exchange
  3. Public Key Cryptosystems

Trusted Server

Basic key exchange,
Needham-Schroeder, Kerberos

Trusted Third-Party

Idea: A central key server delegates keys to everyone:

- The server **T** is trusted by every user on the system
  - Could be either software or dedicated hardware
- Every user has a unique secret key:
  - Client A has key $\mathbf{K_A}$
  - Client B has key $\mathbf{K_B}$, etc.
- **T** knows every user's secret key

Trusted Third Party

The key exchange procedure:

- If client **A** wants to communicate with **B**, then **A** sends a request to server **T**
- **T** generates a random session key ($\mathbf{K_{AB}}$), encrypts it twice (once with K_A and once with K_B), and sends both versions back to **A**
- **A** will decrypt its copy of K_{AB} from the portion that was encrypted with K_A
- **A** will send the other portion to **B**; it was encrypted with K_B, so **B** can decode K_{AB}
- **A** and **B** now share the same secret ($\mathbf{K_{AB}}$) that they can use as a key

Trusted Third-Party

In **security protocol notation**:

1. **A** ➔ **T** : { A, B }
2. **T** ➔ **A** : { K_{AB} }$_{KA}$, { K_{AB} }$_{KB}$
3. **A** ➔ **B** : { K_{AB} }$_{KB}$

Problems with this protocol:

- **B** does not know with whom he's communicating
- A third-party attacker (Eve) could capture the {K_{AB}}$_{KB}$ message, along with any subsequent messages from **A** to **B** (*e.g.*, "send $1M to Eve") and **replay** them later, in order to make B repeat a previous action: **B** can't tell if the message actually came from **A**
- Avoiding such attacks adds considerable complexity

Needham-Schroeder Protocol

1. **A ➔ T** : { A, B, N_A }
   // **A** picks a **nonce** (random number): N_A

2. **T ➔ A** : { N_A, K_{AB}, B, {K_{AB}, A}$_{KB}$ }$_{KA}$
   // N_A in the reply assures **A** that this reply isn't a **replay**
   // Includes **B**'s name: confirms who this is intended for
   // Note that the session key for **B** is encrypted with **A**'s key

3. **A ➔ B** : { K_{AB}, A }$_{KB}$
   // The message from **T** includes **A**'s name

4. **B ➔ A** : { N_B }$_{KAB}$
   // **B** wants to know whether he's actually speaking with **A**
   // Picks his own random nonce: N_B

5. **A ➔ B** : { (N_B – 1) }$_{KAB}$
   // Receiving the result (N_B-1) tells B that A has the key K_{AB}
   // and is responding to new messages (not a replay)

Kerberos

The Needham-Schroeder protocol is essentially how MIT's **Kerberos** system works

- Kerberos is still in use today
- Many other proprietary schemes were also based on similar protocols

Problems with Trusted Server

- Because the system trusts the central server:
  - If **T** is compromised, the attacker has access to every session key as well as every user key
  - An attacker that cannot gain access to **T** can still try to crash the server or overload it, making secure communications impossible
- The system's shortcoming is that the central server represents a single point of failure
- **Diffie-Hellman** and **Public Key Cryptography** (RSA) allow establishing secure communications without a trusted server

Diffie-Hellman Key Exchange

Finite Fields, modular arithmetic, attacks

Diffie-Hellman Key Exchange

This key exchange was the first **public key algorithm**, invented by Whitfield Diffie and Martin Hellman in 1976

- Can be used by two parties to establish a common (short) secret over an insecure link
- Not very efficient for long messages
- Based on the assumption that discrete logarithms (logarithms in modular arithmetic) are difficult to compute

Finite Fields

Both Diffie-Hellman and RSA use **modular arithmetic** operations in a **finite field**:

- A limited set of **n** elements, where (**n** > 1):

  $Z^*{}_p = \{ 0, 1, 2, ... , (\mathbf{n}-1) \}$
- Every element **x** has an additive inverse, such that: $\mathbf{x} + \mathbf{x'} = 0$
- Every element, other than 0, has a multiplicative inverse: $\mathbf{x} \bullet \mathbf{x'} = 1$

Modular Arithmetic

Modular operations are similar to normal operations, except the result is "rounded" by the modulus of **n**:

- **a** mod **n** = remainder(**a**/**n**)
- For any value **a**, the value lies between 0 and (**n**-1)

Say our system has a modulus of 7:

- ([4 + 3] mod 7) = (7 mod 7) = **0**
- ([4 • 3] mod 7) = (12 mod 7) = **5**

Note there are no negative numbers or fractions in modular arithmetic

- If there are no negative numbers or fractions, how do we get additive and multiplicative inverse?

Modular Arithmetic

n																
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1	2
n mod 7																

- Additive inverse of 4? $(\ 4 + \underline{\ 3\ }\ ) \bmod 7 = 0$
- Multiplicative inverse of 4? $(\ 4 \bullet \underline{\ 2\ }\ ) \bmod 7 = 1$
- Multiplicative inverse of 5? $(\ 5 \bullet \underline{\ 3\ }\ ) \bmod 7 = 1$

Modular Arithmetic in Finite Field

Observation: For modular arithmetic to work in a finite field, the modulus must be **prime**

- If the modulus is composite, then some numbers will not have a multiplicative inverse

For example, suppose the modulus is 8: which numbers don't have a multiplicative inverse?

n																
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0
n mod 8																

Exponentiation in Modular Arithmetic

Exponentiation and logarithms in the finite field:

- **Exponentiation:** $(4^3 \bmod 7) = (64 \bmod 7) = 1$
- **Log:** $(\log_4 1)_{\bmod 7} = 3$

What is the discrete log of: $(\log_3 5)_{\bmod 7}$

- Trying to find **x** where: $3^x \bmod 7 = 5$
  - $3^1 = (3 \bmod 7) = 3$
  - $3^2 = (9 \bmod 7) = 2$
  - $3^3 = (27 \bmod 7) = 6$
  - $3^4 = (81 \bmod 7) = 4$
  - **$3^5 = (243 \bmod 7) = 5$**

What is the **complexity** of finding the log?

- Grows with the size of the modulus: NP-hard
- Discrete log is an example of a one-way* function

Diffie-Hellman Algorithm

Initialization:

Alice selects **n**, a **large prime modulus**, as well as a specially selected number **g**, a **generator** of the field n, that lies between between 1 and (**n**-1)

- A number **g** is a generator of field **n** if, for each **y** between 1 and (**n**-1), there exists an **x** such that $g^x \bmod n = y$
  - { $g^0, g^1, g^2, g^3, \ldots, g^{n-1}$ } yields all numbers from { 1 … (n-1) }
- For more details on generator selection, refer to the *Handbook of Applied Cryptography*

Diffie-Hellman Key Exchange

Establishing a shared secret:

- Alice selects a random integer **x** and computes: **P = g^x mod n**
- Alice sends **P**, **g** and **n** to Bob; **x** is kept secret
- Bob selects a random integer **y** and computes: **Q = g^y mod n**
- Bob sends **Q** back to Alice; **y** is kept secret
- Alice computes **Q^x mod n** = g^{xy} mod n
- Bob also computes **P^y mod n** = g^{xy} mod n

They now both share the secret **g^{xy} mod n**

Attacking Diffie-Hellman

Suppose Eve is listening to Alice and Bob:

- She only sees the values **P**, **Q**, **g** and **n**
- She doesn't know **x** or **y**
- She must solve a discrete log to discover the secret value $\mathbf{g^{xy} \bmod n}$

The *Man-in-the-Middle* Attack

With Diffie-Hellman, Alice does not know whether she is performing the key exchange with Bob or Eve: the algorithm is vulnerable to a **man-in-the-middle attack**

- If Eve can pretend to be Bob when communicating with Alice, and pretend to be Alice when communicating with Bob, she can eavesdrop on their communications
- Eve can establish a shared secret with each, decrypt the message from one, encrypt it for other, pass it along
- This is a problem if Eve has control of a router along the communication path between Alice and Bob

Shared Secret 1 — Shared Secret 2

Alice ⇄ Eve ⇄ Bob

The *Man-in-the-Middle* Attack

- The problem with Diffie-Hellman is that it does not authenticate (identify) the remote party
- Next we look at Public-Key based key exchange that allows a user to create a message that can only be decrypted by the intended recipient

Public-Key Key Exchange

Introduction

Public Key Cryptosystems

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

- Creates an **asymmetric** cryptosystem
- 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

- During 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

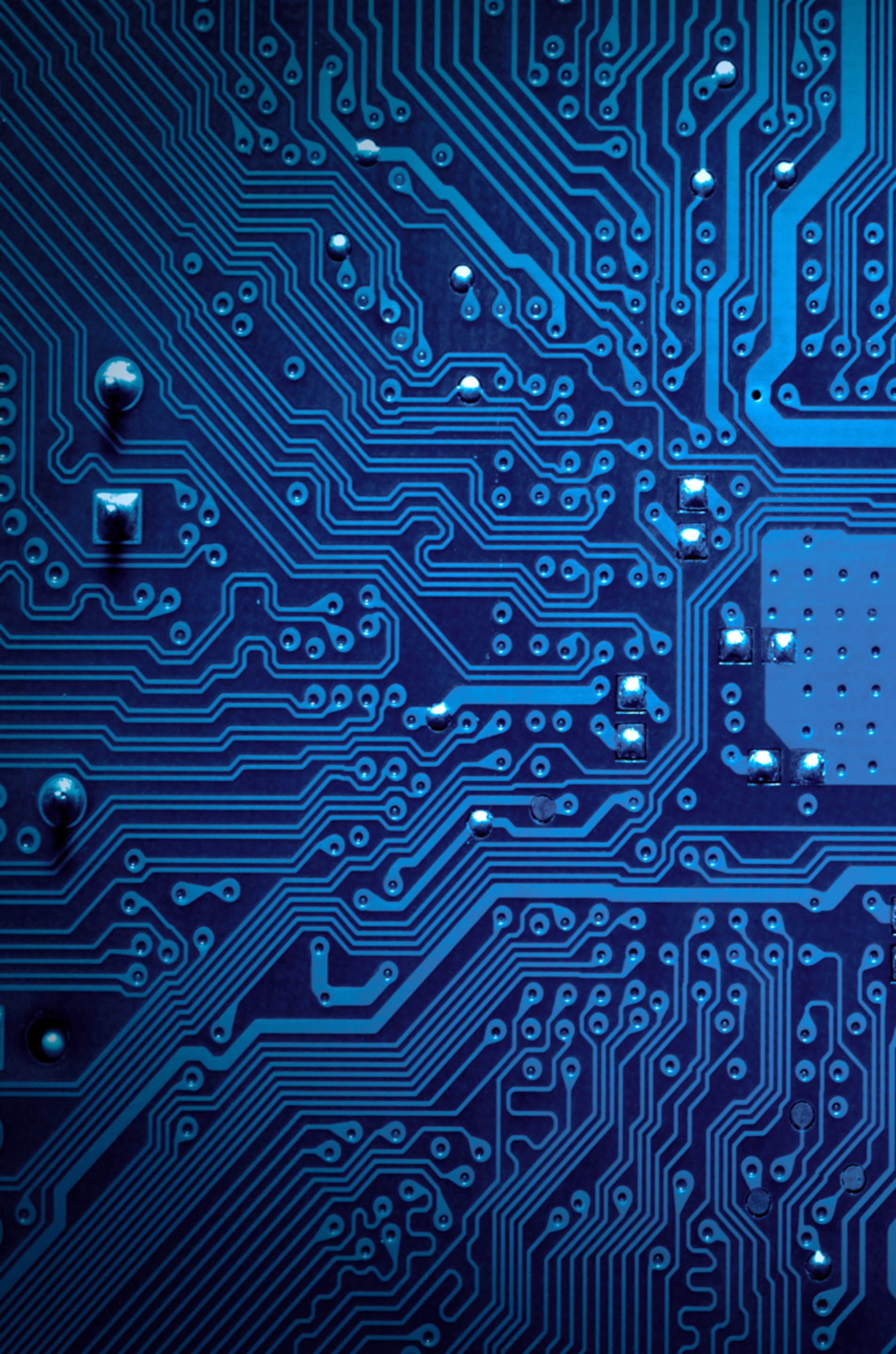
Public-Key Exchange of Keys

Setting up a shared secret using a public key cryptosystem is straight-forward:

- Alice randomly selects a key **x** and encrypts it with Bob's public key
- Bob receives the encrypted key and decrypts it with his private key
- Both Alice and Bob now share the same key **x**
- Is there any problem with this scheme?

Two popular public key algorithms are RSA and DSA

- RSA is based on factoring
- Digital Signature Algorithm (DSA) is based on discrete logs and uses the same principle as Diffie-Hellman



Questions?