

SE 4472

Information Security

Week 7

Public-key Cryptography

Aleksander Essex



Symmetric-key cryptography

In terms of cryptographic primitives, we've covered ciphers (block ciphers, stream ciphers), message authentication codes (MACs), and hashes. Hashes don't require a key, but ciphers and MACs each require a *secret* key.

- ▶ The secret key k is used to "do" something (i.e., encrypt, resp. MAC)
- ▶ The secret key k is also used to "undo" that something (i.e., decrypt, resp. verify MAC)
- ▶ Called *symmetric*-key because the *do* key is the same as the *undo* key

Symmetric-key cryptography

In our Alice/Bob/Eve communication model, who knows the secret key k ?

- ▶ Alice knows k
- ▶ Bob knows k
- ▶ Eve does not know k
- ▶ It should be computationally infeasible to guess k

The Million Dollar Question

Suppose Alice wants to communicate privately with Bob. She could generate a key and encrypt her message with a block cipher. Bob, however, will require this key to be able to decrypt her message.

Question: How does Alice communicate the secret key k to Bob while keeping it secret from Eve?

The Million Dollar Question

Suppose Alice wants to communicate privately with Bob. She could generate a key and encrypt her message with a block cipher. Bob, however, will require this key to be able to decrypt her message.

Question: How does Alice communicate the secret key k to Bob while keeping it secret from Eve?

The Goal

Alice and Bob want to communicate privately. They need to agree on a shared secret (a key), but only have an insecure network over which to communicate.

Merkle Puzzles: A thought experiment

Ralph Merkle considered this problem as an undergrad class project in 1974 and had the following solution:

Merkle Puzzles: A thought experiment

Ralph Merkle considered this problem as an undergrad class project in 1974 and had the following solution:

1. Alice generates a puzzle that is moderately difficult to solve. The solution to the puzzle reveals an independent random secret key k and an ID

Merkle Puzzles: A thought experiment

Ralph Merkle considered this problem as an undergrad class project in 1974 and had the following solution:

1. Alice generates a puzzle that is moderately difficult to solve. The solution to the puzzle reveals an independent random secret key k and an ID
2. Alice repeats step 1 to generate a set of n puzzles $p_1 \dots p_n$ corresponding to keys $k_1 \dots k_n$ and associated IDs $d_1 \dots d_n$

Merkle Puzzles: A thought experiment

Ralph Merkle considered this problem as an undergrad class project in 1974 and had the following solution:

1. Alice generates a puzzle that is moderately difficult to solve. The solution to the puzzle reveals an independent random secret key k and an ID
2. Alice repeats step 1 to generate a set of n puzzles $p_1 \dots p_n$ corresponding to keys $k_1 \dots k_n$ and associated IDs $d_1 \dots d_n$
3. She sends the puzzle set to Bob. He picks puzzle p_i at random and solves it, revealing key k_i and ID d_i

Merkle Puzzles: A thought experiment

Ralph Merkle considered this problem as an undergrad class project in 1974 and had the following solution:

1. Alice generates a puzzle that is moderately difficult to solve. The solution to the puzzle reveals an independent random secret key k and an ID
2. Alice repeats step 1 to generate a set of n puzzles $p_1 \dots p_n$ corresponding to keys $k_1 \dots k_n$ and associated IDs $d_1 \dots d_n$
3. She sends the puzzle set to Bob. He picks puzzle p_i at random and solves it, revealing key k_i and ID d_i
4. Bob sends ID d_i to Alice

Merkle Puzzles: A thought experiment

Ralph Merkle considered this problem as an undergrad class project in 1974 and had the following solution:

1. Alice generates a puzzle that is moderately difficult to solve. The solution to the puzzle reveals an independent random secret key k and an ID
2. Alice repeats step 1 to generate a set of n puzzles $p_1 \dots p_n$ corresponding to keys $k_1 \dots k_n$ and associated IDs $d_1 \dots d_n$
3. She sends the puzzle set to Bob. He picks puzzle p_i at random and solves it, revealing key k_i and ID d_i
4. Bob sends ID d_i to Alice
5. Alice and Bob begin communicating securely using key k_i

Merkle Puzzles: A thought experiment

Ralph Merkle considered this problem as an undergrad class project in 1974 and had the following solution:

1. Alice generates a puzzle that is moderately difficult to solve. The solution to the puzzle reveals an independent random secret key k and an ID
2. Alice repeats step 1 to generate a set of n puzzles $p_1 \dots p_n$ corresponding to keys $k_1 \dots k_n$ and associated IDs $d_1 \dots d_n$
3. She sends the puzzle set to Bob. He picks puzzle p_i at random and solves it, revealing key k_i and ID d_i
4. Bob sends ID d_i to Alice
5. Alice and Bob begin communicating securely using key k_i

Questions: How does this prevent Eve from guessing k_i ? Is this approach practical?

Asymmetric-key cryptography

Let's now consider a new class of cryptographic primitive. This primitive will make use of two distinct keys: a *public* key PU and a *private* key PR .

- ▶ The public key PU is used to "do" something (e.g., encrypt)
- ▶ The private key PR is used to "undo" that something (e.g., decrypt)
- ▶ Called *asymmetric*-key because the *do* key is different from *undo* key

Asymmetric-key cryptography

In our Alice/Bob/Eve communication model, who knows the public and private keys PU , PR ?

- ▶ Alice knows the public key PU
- ▶ Bob knows the public and private PU , PR
- ▶ Eve knows the public key
- ▶ Anyone and everyone can know the public key
- ▶ It should be computationally infeasible to guess the private key
- ▶ It should be computationally infeasible to recover the private key given the public key

Terminology

- ▶ Asymmetric-key crypto systems are more commonly called "Public-key" systems.
- ▶ The public and private keys are jointly referred to as a *key pair*.
- ▶ Use *secret key* when talking about the key of a symmetric-key cryptosystem
- ▶ Use *private key* when talking about the secret/private key of a asymmetric-key cryptosystem

Applications of Public-key Cryptography

- ▶ Encryption/decryption (e.g., RSA)
- ▶ Key agreement/exchange (e.g., Diffie-Hellman: DHE, ECDHE, etc)
- ▶ Digital signatures (e.g., RSA, Elgamal, DSA, ECDSA, etc)
- ▶ Advanced: secure/homomorphic computation, zero-knowledge proofs, etc

Discrete Logarithms: math primer

Let

- ▶ p, q be prime numbers such that $p = \alpha q + 1$ for some integer α
- ▶ \mathbb{Z}_n denote the set of integers modulo an integer n
- ▶ the multiplicative inverse of a number $a \in \mathbb{Z}_n$, denoted a^{-1} , be an integer in \mathbb{Z}_n such that $aa^{-1} = 1 \pmod n$
- ▶ \mathbb{Z}_n^* be the set of integers modulo n for which a multiplicative inverse exists. If n is prime, $\mathbb{Z}_p = \{1, 2, \dots, n - 1\}$

Discrete Logarithms: math primer

Let

- ▶ a be an element in \mathbb{Z}_p^* . Let t be the smallest integer such that $a^t = 1 \pmod p$. We call t the *order* of a . If $t = p$ then we say a is a generator of \mathbb{Z}_p^* .
- ▶ a is called a *generator* because the set $\{a^0, a^1 \dots a^{p-2}\} = \{1, \dots, p-1\}$, i.e., a generates the set \mathbb{Z}_p^* . For example, 6 generates \mathbb{Z}_{13}^* since $\{6^1 = 6 \pmod{13}, 6^2 = 10 \pmod{13}, 6^3 = 8 \pmod{13}, \dots, 6^{12} = 1 \pmod{13}\}$
- ▶ \mathbb{G}_q denote a cyclic subgroup of \mathbb{Z}_p^* of order q . An element $a \in \mathbb{G}_q$ is also an element $a \in \mathbb{Z}_p^*$. Let g be a generator of \mathbb{G}_q
- ▶ $a \in_R A$ denote an element a drawn independently and uniformly at random from set A . $a \in_R \mathbb{G}_q$, therefore, would denote a random element in \mathbb{G}_q

\mathbb{G}_q : an example

Let $q = 11$ and $p = 23 = 2 * q + 1$. Let $g = 6$.

$$6^1 \pmod{23} = 6$$

$$6^2 \pmod{23} = 13$$

$$6^3 \pmod{23} = 9$$

$$6^4 \pmod{23} = 8$$

$$6^5 \pmod{23} = 2$$

$$6^6 \pmod{23} = 12$$

$$6^7 \pmod{23} = 3$$

$$6^8 \pmod{23} = 18$$

$$6^9 \pmod{23} = 16$$

$$6^{10} \pmod{23} = 4$$

$$6^{11} \pmod{23} = 1$$

$$6^{12} \pmod{23} = 6$$

$$6^{13} \pmod{23} = 13$$

⋮

G_q : a full-scale example

2048-bit modulus. Let:

$p =$
1699897197819409959350395909560868339296707333513338850260792175269377461667909345106189
4007390651442940991437007217396778219812942355822485419132091732942087052688780401771105
5077916007496804049206725568956610515399196848621653907978580213217522397058071043503404
700268425750722626265208099856407306527012763

$q =$
8499485989097049796751979547804341696483536667566694251303960876346887308339546725530947
0036953257214704957185036086983891099064711779112427095660458664710435263443902008855527
5389580037484020246033627844783052576995984243108269539892901066087611985290355217517023
50134212875361313132604049928203653263506381

$g =$
6811145128679259384514506369165999341022181280687423436585450471905740185837259494289329
1581957322023471947260828209362467690671421429979048643907159864269436501403220400197614
3089044605475295746938752186625055539386825735547196324910243046376438686033381140427605
29545510633271426088675581644231528918421974

G_q : a full-scale example

Try it for yourself in a Python command line:

```
>>> g**2%p
```

```
7868561465852767168535877469150839014533636907794303563749025200723864470735156596654449  
9964565109242525722680966879319045332149033624253533553705188800820306775346364385456081  
9837040305333253946350048768241942227777332942718344921601872312324632112234290209388173  
87400162156412030140010622964762147232938172
```

```
>>> g**3%p =
```

```
9330114779345580939484179789597298690260403750404885274182886474168552330554964253269877  
7805035962128777511734695474769570333684953304250729699841923239590754869761035510310467  
3380434003110925615598887252074369149132484160888674079252315630580538390956978412596567  
42157441575413639958516702704106360282167237
```

```
>>> g**(q-1)%p
```

```
3367616480157963046716355041037478200291764055973664780592326569211350886084056857190221  
5616086225978249457926967010402533726919077542155914501602289719125842308243545002368037  
4701884700401430621838112321925898253632244044710276421178889345050342849101359479604575  
93982808718223408019267225489553349654893967
```

```
>>> g**q%p
```

```
1
```

Note: Python is not optimized to do big exponentiations, so computing g^q is going to take a while

The Discrete Logarithm Problem

Suppose I gave you p , q , g and the following value:

$$a = g^r \pmod p =$$

77615165225041151606667206153311570843582939776216781720406344828508303008415498392953
29074916151320052641461615321304724851857182369013464691908418673868090067406047464217
91799479531358291540981935563613210692823031038873030722308677544198575890762028642253
55598512052408316495848979053582277338958932302286

Could you tell me what r was?

The Discrete Logarithm Problem

Suppose I gave you p , q , g and the following value:

$$a = g^r \pmod{p} =$$

77615165225041151606667206153311570843582939776216781720406344828508303008415498392953
29074916151320052641461615321304724851857182369013464691908418673868090067406047464217
91799479531358291540981935563613210692823031038873030722308677544198575890762028642253
55598512052408316495848979053582277338958932302286

Could you tell me what r was?

Short answer: No, you can't, or more specifically, it would be computationally infeasible to do so. This is the *discrete log* problem, and the infeasibility of solving it is an important computational hardness assumption that we can use to build a key agreement protocol.

The Discrete Logarithm Problem

The discrete logarithm problem (DLP) can be stated as follows: let \mathbb{G}_q be a cyclic subgroup of \mathbb{Z}_p^* of order q with generator g .

Given $\langle p, q, g \rangle$ and a value $a = g^b \pmod p$ for some $b \in \mathbb{Z}_q$, determine b .

The Diffie-Hellman key agreement protocol

- ▶ Proposed by Whitfield Diffie and Martin Hellman in 1976
- ▶ Uses the discrete logarithm problem to generate a public key $PU_A = g^a$ from a private key $PR_A = a$
- ▶ Uses the commutative nature of exponentiation:
 $(g^a)^b = (g^b)^a = g^{ab}$

The Diffie-Hellman key agreement protocol

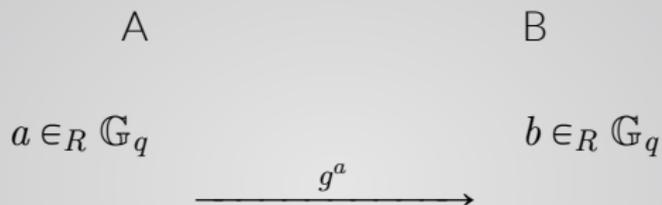
A

$$a \in_R \mathbb{G}_q$$

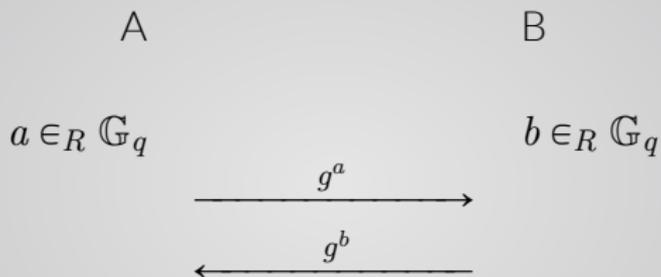
B

$$b \in_R \mathbb{G}_q$$

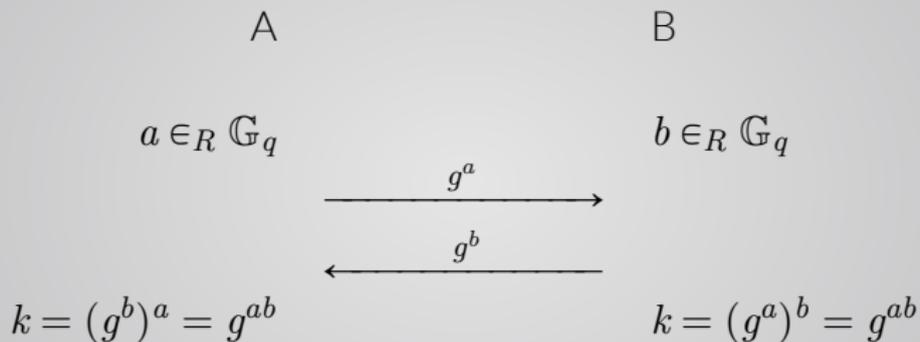
The Diffie-Hellman key agreement protocol



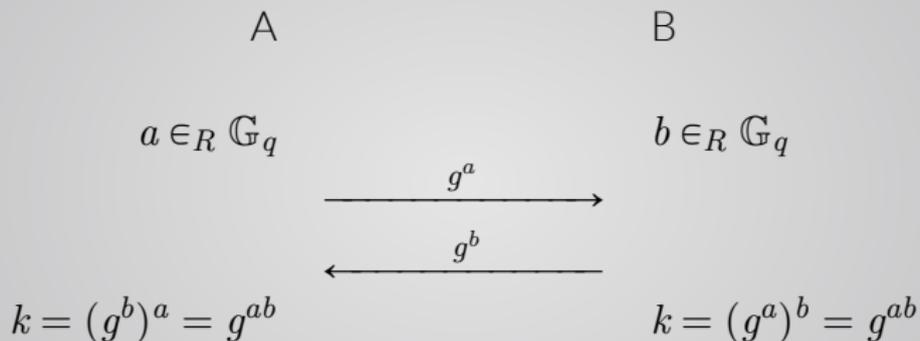
The Diffie-Hellman key agreement protocol



The Diffie-Hellman key agreement protocol



The Diffie-Hellman key agreement protocol



Exercise: what values are public keys, private keys and secret keys?

The Diffie-Hellman Problem

The Diffie-Hellman problem (DHP), stated as follows:

Given g, g^a, g^b compute g^{ab} . This is assumed to be hard if the DL problem is hard.

The Diffie-Hellman Problem

The Diffie-Hellman problem (DHP), stated as follows:

Given g, g^a, g^b compute g^{ab} . This is assumed to be hard if the DL problem is hard.

The collection of values $\langle g, g^a, g^b, g^{ab} \rangle$ is called a Diffie Hellman *tuple*. Another useful assumption is the Decisional Diffie Hellman assumption (DDH), stated as follows:

The Diffie-Hellman Problem

The Diffie-Hellman problem (DHP), stated as follows:

Given g, g^a, g^b compute g^{ab} . This is assumed to be hard if the DL problem is hard.

The collection of values $\langle g, g^a, g^b, g^{ab} \rangle$ is called a Diffie Hellman *tuple*. Another useful assumption is the Decisional Diffie Hellman assumption (DDH), stated as follows:

Given $\langle g, g^a, g^b, g^{ab} \rangle$ and $\langle g, g^a, g^b, g^c \rangle$ for $a, b, c \in_R \mathbb{Z}_q$, decide which is the valid Diffie Hellman tuple. This is assumed to be hard if $g \in \mathbb{G}_q$ and the DLP is hard in \mathbb{G}_q . DDH is useful for proving the CPA-security of cryptosystems based on DLP.

Man-in-the-Middle Attacks

Diffie-Hellman is **highly** susceptible to man-in-the-middle attacks. The goal of the attack is for Eve to be able to eavesdrop on Alice and Bob. The attack proceeds as follows:

Man-in-the-Middle Attacks

Diffie-Hellman is **highly** susceptible to man-in-the-middle attacks. The goal of the attack is for Eve to be able to eavesdrop on Alice and Bob. The attack proceeds as follows:

1. Alice (unwittingly) initiates a DH key agreement with Eve who is intercepting messages to Bob. Let k_{AE} be Alice and Eve's shared secret

Man-in-the-Middle Attacks

Diffie-Hellman is **highly** susceptible to man-in-the-middle attacks. The goal of the attack is for Eve to be able to eavesdrop on Alice and Bob. The attack proceeds as follows:

1. Alice (unwittingly) initiates a DH key agreement with Eve who is intercepting messages to Bob. Let k_{AE} be Alice and Eve's shared secret
2. Eve initiates a DH key agreement with Bob (who is now impersonating Alice). Let k_{EB} be Eve and Bob's shared secret

Man-in-the-Middle Attacks

Diffie-Hellman is **highly** susceptible to man-in-the-middle attacks. The goal of the attack is for Eve to be able to eavesdrop on Alice and Bob. The attack proceeds as follows:

1. Alice (unwittingly) initiates a DH key agreement with Eve who is intercepting messages to Bob. Let k_{AE} be Alice and Eve's shared secret
2. Eve initiates a DH key agreement with Bob (who is now impersonating Alice). Let k_{EB} be Eve and Bob's shared secret
3. When Alice sends a message to Bob, she encrypts it with k_{AE} . Eve intercepts the message, decrypts it, and then re-encrypts it with k_{EB} and forwards to Bob

Man-in-the-Middle Attacks

Diffie-Hellman is **highly** susceptible to man-in-the-middle attacks. The goal of the attack is for Eve to be able to eavesdrop on Alice and Bob. The attack proceeds as follows:

1. Alice (unwittingly) initiates a DH key agreement with Eve who is intercepting messages to Bob. Let k_{AE} be Alice and Eve's shared secret
2. Eve initiates a DH key agreement with Bob (who is now impersonating Alice). Let k_{EB} be Eve and Bob's shared secret
3. When Alice sends a message to Bob, she encrypts it with k_{AE} . Eve intercepts the message, decrypts it, and then re-encrypts it with k_{EB} and forwards to Bob
4. Eve applies the same strategy in reverse when Bob sends a message to Alice

Conclusion

1. Diffie-Hellman was an extremely important discovery: now two parties who have never met can exchange messages over an insecure network to arrive at a shared secret
2. Widely used by TLS (Diffie-Hellman key exchange, or DHE)
3. MITM attacks are a real-world threat. You *need* to know who you are talking to.