Cryptography
Week #9:
Diffie-Hellman and the discrete lograrithm problem

Rogério Reis, rogerio.reis@fc.up.pt
2024/2025
DCC FCUP

November, 23rd 2024

Discrete Logarithm Problem (DLP)

Let a and n be integers such that $(a, n) = 1$. We know, by Euler's Theorem that the equation

$$a^m \equiv 1 \pmod{n},$$

has, at least, one solution (making $m = \phi(n)$).

Discrete Logarithm Problem (DLP)

Let a and n be integers such that $(a, n) = 1$. We know, by Euler's Theorem that the equation

$$a^m \equiv 1 \pmod{n},$$

has, at least, one solution (making $m = \phi(n)$).

Definition

Let a and n be such that $(a, n) = 1$. The smallest positive integer m such that

$$a^m \equiv 1 \pmod{n},$$

is called **the order of** $a \pmod{n}$.

Definition (Primitive root)

Let a and n be such that $(a, n) = 1$, a is called a **primitive root** of n if the order of $a \pmod{n}$ is $\phi(n)$.

Definition (Primitive root)

Let a and n be such that $(a, n) = 1$, a is called a **primitive root** of n if the order of $a \pmod{n}$ is $\phi(n)$.

Not all integers have primitive roots. In fact, only the following integers have primitive roots: $2, 4, p^n$ and $2p^n$ with p odd prime.

Powers in $\mathbb{Z}_{19}^*$

a^1	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	**4**	**8**	**16**	**13**	**7**	**14**	**9**	**18**	**17**	**15**	**11**	**3**	**6**	**12**	**5**	**10**	**1**
3	**9**	**8**	**5**	**15**	**7**	**2**	**6**	**18**	**16**	**10**	**11**	**14**	**4**	**12**	**17**	**13**	**1**
4	**16**	**7**	**9**	**17**	**11**	**6**	**5**	**1**	4	16	7	9	17	11	6	5	1
5	**6**	**11**	**17**	**9**	**7**	**16**	**4**	**1**	5	6	11	17	9	7	16	4	1
6	**17**	**7**	**4**	**5**	**11**	**9**	**16**	**1**	6	17	7	4	5	11	9	16	1
7	**11**	**1**	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	**7**	**18**	**11**	**12**	**1**	8	7	18	11	12	1	8	7	18	11	12	1
9	**5**	**7**	**6**	**16**	**11**	**4**	**17**	**1**	9	5	7	6	16	11	4	17	1
10	**5**	**12**	**6**	**3**	**11**	**15**	**17**	**18**	**9**	**14**	**7**	**13**	**16**	**8**	**4**	**2**	**1**
11	**7**	**1**	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	**11**	**18**	**7**	**8**	**1**	12	11	18	7	8	1	12	11	18	7	8	1
13	**17**	**12**	**4**	**14**	**11**	**10**	**16**	**18**	**6**	**2**	**7**	**15**	**6**	**8**	**9**	**3**	**1**
14	**6**	**8**	**17**	**10**	**7**	**3**	**4**	**18**	**5**	**13**	**11**	**2**	**9**	**12**	**16**	**15**	**1**
15	**16**	**12**	**9**	**2**	**11**	**13**	**5**	**18**	**4**	**3**	**7**	**10**	**17**	**8**	**6**	**14**	**1**
16	**9**	**11**	**5**	**4**	**7**	**17**	**6**	**1**	16	9	11	5	4	7	17	6	1
17	**4**	**11**	**16**	**6**	**7**	**5**	**9**	**1**	17	4	11	16	6	7	5	9	1
18	**1**	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1

Let p be a prime integer, and a a primitive root of p. Then we know that the powers of a (from 1 to $(p-1)$) are all distinct, and thus "cover" all integers from 1 to $(p-1)$.

Let p be a prime integer, and a a primitive root of p. Then we know that the powers of a (from 1 to $(p-1)$) are all distinct, and thus "cover" all integers from 1 to $(p-1)$.Hence as for any integer b there exists a r such that

$$b \equiv r \pmod{p} \text{ (with } 0 \leq r \leq (p-1)),$$

there must exist a sole power i such that

$$b \equiv a^i \pmod{p} \text{ (with } 0 \leq i \leq (p-1) \text{).}$$

Definition

Let a and p, with p prime and a primitive root of p. Let b be an integer, i is called **index of** b with respect to $a \pmod{p}$ if

$$b \equiv a^i \pmod{p}.$$

Note that one has:

$$ind_{a,p}(1) \quad = \quad 0, \text{ because } a^0 \mod p = 1 \mod p = 1$$

Note that one has:

$$\begin{aligned} ind_{a,p}(1) &= 0, \text{ because } a^0 \mod p = 1 \mod p = 1 \\ ind_{a,p}(a) &= 1, \text{ because } a^1 \mod p = a. \end{aligned}$$

Note that one has:

$$
\begin{aligned}
ind_{a,p}(1) &= 0, \text{ because } a^0 \mod p = 1 \mod p = 1 \\
ind_{a,p}(a) &= 1, \text{ because } a^1 \mod p = a.
\end{aligned}
$$

$$
\begin{aligned}
xy &= a^{ind_{a,p}(xy)} \mod p \\
&= (a^{ind_{a,p}(x)} \mod p)(a^{ind_{a,p}(y)} \mod p) \\
&= (a^{ind_{a,p}(x)+ind_{a,p}(y)} \mod p)
\end{aligned}
$$

Note that one has:

$$\begin{aligned} ind_{a,p}(1) &= 0, \text{ because } a^0 \mod p = 1 \mod p = 1 \\ ind_{a,p}(a) &= 1, \text{ because } a^1 \mod p = a. \end{aligned}$$

$$\begin{aligned} xy &= a^{ind_{a,p}(xy)} \mod p \\ &= (a^{ind_{a,p}(x)} \mod p)(a^{ind_{a,p}(y)} \mod p) \\ &= (a^{ind_{a,p}(x)+ind_{a,p}(y)} \mod p) \end{aligned}$$

$$ind_{a,p}(x) \qquad \log_a(x) \pmod p$$

Diffie-Hellman basics

For a given pair of agents, Alice (A) and Bob (B), they need to agree on a public multiplicative group $\mathbb{Z}_p^*$,

Diffie-Hellman basics

For a given pair of agents, Alice (A) and Bob (B), they need to agree on a public multiplicative group $\mathbb{Z}_p^*$,i.e. they need to agree in a prime p, and a base $g \in \mathbb{Z}_p^*$.

Diffie-Hellman basics

For a given pair of agents, Alice (A) and Bob (B), they need to agree on a public multiplicative group $\mathbb{Z}_p^*$,i.e. they need to agree in a prime p, and a base $g \in \mathbb{Z}_p^*$.

Alice and Bob generate random private keys, i.e. integers $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_p^*$,

Diffie-Hellman basics

For a given pair of agents, Alice (A) and Bob (B), they need to agree on a public multiplicative group $\mathbb{Z}_p^*$,i.e. they need to agree in a prime p, and a base $g \in \mathbb{Z}_p^*$.

Alice and Bob generate random private keys, i.e. integers $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_p^*$,and generate their public keys: $A = g^a \mod p$ and $B = g^b \mod p$.

Diffie-Hellman basics

For a given pair of agents, Alice (A) and Bob (B), they need to agree on a public multiplicative group $\mathbb{Z}_p^*$,i.e. they need to agree in a prime p, and a base $g \in \mathbb{Z}_p^*$.

Alice and Bob generate random private keys, i.e. integers $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_p^*$,and generate their public keys: $A = g^a \mod p$ and $B = g^b \mod p$.

Now, if they exchange their respective public keys (through a public unprotected channel), A and B, they can agree in a secret value without the need ever transmitting it:

$$k = A^b = (g^a)^b = g^{ab} = (g^b)^a = B^a.$$

Diffie-Hellman basics

For a given pair of agents, Alice (A) and Bob (B), they need to agree on a public multiplicative group $\mathbb{Z}_p^*$,i.e. they need to agree in a prime p, and a base $g \in \mathbb{Z}_p^*$.

Alice and Bob generate random private keys, i.e. integers $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_p^*$,and generate their public keys: $A = g^a \mod p$ and $B = g^b \mod p$.

Now, if they exchange their respective public keys (through a public unprotected channel), A and B, they can agree in a secret value without the need ever transmitting it:

$$k = A^b = (g^a)^b = g^{ab} = (g^b)^a = B^a.$$

They share, now, a secret k.

The resulting value, $k = g^{ab}$, is the shared secret; it is then passed to a key derivation function (KDF) in order to generate one or more shared symmetric keys. A KDF is a kind of hash function that will return a random-looking string the size of the desired key length.

The resulting value, $k = g^{ab}$, is the shared secret; it is then passed to a key derivation function (KDF) in order to generate one or more shared symmetric keys. A KDF is a kind of hash function that will return a random-looking string the size of the desired key length.

To enhance security of the system g should be a primitive root of p, if not, only a small number of values can be generated as the shared secret.

The resulting value, $k = g^{ab}$, is the shared secret; it is then passed to a key derivation function (KDF) in order to generate one or more shared symmetric keys. A KDF is a kind of hash function that will return a random-looking string the size of the desired key length.

To enhance security of the system g should be a primitive root of p, if not, only a small number of values can be generated as the shared secret.

To ensure the highest security, safe DH parameters should work with a prime p such that $(p-1)/2$ is also prime. Such a safe prime guarantees that the group doesn't have small subgroups that would make DH easier to break. With a safe prime, DH can notably work with $g = 2$, which makes computations slightly faster. But generating safe prime p a takes more time than generating a totally random prime.

The resulting value, $k = g^{ab}$, is the shared secret; it is then passed to a key derivation function (KDF) in order to generate one or more shared symmetric keys. A KDF is a kind of hash function that will return a random-looking string the size of the desired key length.

To enhance security of the system g should be a primitive root of p, if not, only a small number of values can be generated as the shared secret.

To ensure the highest security, safe DH parameters should work with a prime p such that $(p-1)/2$ is also prime. Such a safe prime guarantees that the group doesn't have small subgroups that would make DH easier to break. With a safe prime, DH can notably work with $g = 2$, which makes computations slightly faster. But generating safe prime p a takes more time than generating a totally random prime.Generating DH parameters is about 1000 slower than generating RSA parameters, for the same security level.

The Computational Diffie-Hellman Problem

The computational Diffie-Hellman (CDH) problem is that of computing the shared secret g^{ab} given only the public values g^a and g^b, and not any of the secret values a or b.

The Computational Diffie-Hellman Problem

The computational Diffie-Hellman (CDH) problem is that of computing the shared secret g^{ab} given only the public values g^a and g^b, and not any of the secret values a or b.That is, if someone can get the secret g^{ab} with only the information that is available in the public channel.

The Computational Diffie-Hellman Problem

The computational Diffie-Hellman (CDH) problem is that of computing the shared secret g^{ab} given only the public values g^a and g^b, and not any of the secret values a or b.That is, if someone can get the secret g^{ab} with only the information that is available in the public channel. We know that DLP is at least as hard as CDH but we do not know how to prove the equivalence.

The Computational Diffie-Hellman Problem

The computational Diffie-Hellman (*CDH*) problem is that of computing the shared secret g^{ab} given only the public values g^a and g^b, and not any of the secret values a or b.That is, if someone can get the secret g^{ab} with only the information that is available in the public channel. We know that *DLP* is at least as hard as *CDH* but we do not know how to prove the equivalence.

Diffie-Hellman shares another similarity with *RSA* in that *DH* will deliver the same security level as RSA for a given modulus size.

The Computational Diffie-Hellman Problem

The computational Diffie-Hellman (*CDH*) problem is that of computing the shared secret g^{ab} given only the public values g^a and g^b, and not any of the secret values a or b.That is, if someone can get the secret g^{ab} with only the information that is available in the public channel. We know that *DLP* is at least as hard as *CDH* but we do not know how to prove the equivalence.

Diffie-Hellman shares another similarity with *RSA* in that *DH* will deliver the same security level as RSA for a given modulus size. Indeed, the fastest way we know to break *CDH* is to solve *DLP* using an algorithm called the number field sieve, a method similar but not identical to the fastest one that breaks *RSA* by factoring its modulus: the general number field sieve (*GNFS*).

The Decisional Diffie-Hellman Problem (*DDH*)

Imagine that an attacker can compute the first 32 bits of g^{ab} given the 2048-bit values of g^a and g^b, but that they cannot compute all 2048 bits. Although *CDH* would still be unbroken because 32 bits aren't enough to completely recover g^{ab}, the attacker would still have learned something about the shared secret, which might still allow them to compromise an application's security.

The Decisional Diffie-Hellman Problem (*DDH*)

Imagine that an attacker can compute the first 32 bits of g^{ab} given the 2048-bit values of g^a and g^b, but that they cannot compute all 2048 bits. Although *CDH* would still be unbroken because 32 bits aren't enough to completely recover g^{ab}, the attacker would still have learned something about the shared secret, which might still allow them to compromise an application's security.

To ensure that an attacker can't learn anything about the shared secret g^{ab}, this value needs only to be indistinguishable from a random group element, just as an encryption scheme is secure when ciphertexts are indistinguishable from random strings.

The Decisional Diffie-Hellman Problem (*DDH*)

Imagine that an attacker can compute the first 32 bits of g^{ab} given the 2048-bit values of g^a and g^b, but that they cannot compute all 2048 bits. Although *CDH* would still be unbroken because 32 bits aren't enough to completely recover g^{ab}, the attacker would still have learned something about the shared secret, which might still allow them to compromise an application's security.

To ensure that an attacker can't learn anything about the shared secret g^{ab}, this value needs only to be indistinguishable from a random group element, just as an encryption scheme is secure when ciphertexts are indistinguishable from random strings.The computational problem formalizing this intuition is called the decisional Diffie-Hellman (*DDH*) problem. Given g^a, g^b, and value that is either g^{ab} or g^c for some random c (each of the two with a chance of $\frac{1}{2}$), the DDH problem consists of determining whether k was chosen.

The Decisional Diffie-Hellman Problem (*DDH*)

Imagine that an attacker can compute the first 32 bits of g^{ab} given the 2048-bit values of g^a and g^b, but that they cannot compute all 2048 bits. Although *CDH* would still be unbroken because 32 bits aren't enough to completely recover g^{ab}, the attacker would still have learned something about the shared secret, which might still allow them to compromise an application's security.

To ensure that an attacker can't learn anything about the shared secret g^{ab}, this value needs only to be indistinguishable from a random group element, just as an encryption scheme is secure when ciphertexts are indistinguishable from random strings.The computational problem formalizing this intuition is called the decisional Diffie-Hellman (*DDH*) problem. Given g^a, g^b, and value that is either g^{ab} or g^c for some random c (each of the two with a chance of $\frac{1}{2}$), the DDH problem consists of determining whether k was chosen.The assumption that no attacker can solve *DDH* efficiently is called the *decisional Diffie-Hellman assumption*.

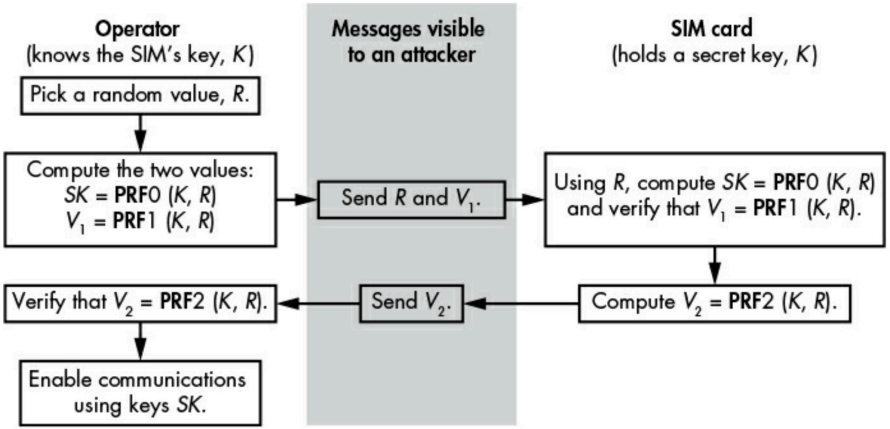
If *DDH* is hard, then *CDH* is also hard, and you can't learn anything about k. But if you can solve *CDH*, you can also solve *DDH*.

If *DDH* is hard, then *CDH* is also hard, and you can't learn anything about k. But if you can solve *CDH*, you can also solve *DDH.*The bottom line is that *DDH* is fundamentally less hard than *CDH*, yet *DDH* hardness is a prime assumption in cryptography, and one of the most studied.

If *DDH* is hard, then *CDH* is also hard, and you can't learn anything about k. But if you can solve *CDH*, you can also solve *DDH.*The bottom line is that *DDH* is fundamentally less hard than *CDH*, yet *DDH* hardness is a prime assumption in cryptography, and one of the most studied.

We can be confident that both CDH and DDH are hard when Diffie-Hellman parameters are well chosen.

An example of non-*DH* key agreement



This protocol is not immune to all kinds of attacks: in principle there's a way to fool the SIM card with replay attack. Essentially, if an attacker captures a pair (R, V_1), they may send it to the SIM card and trick the SIM into believing that the pair came from a legitimate operator that knows K. To prevent this attack, the protocol includes additional checks to ensure that the same R isn't reused.

This protocol is not immune to all kinds of attacks: in principle there's a way to fool the SIM card with replay attack. Essentially, if an attacker captures a pair (R, V_1), they may send it to the SIM card and trick the SIM into believing that the pair came from a legitimate operator that knows K. To prevent this attack, the protocol includes additional checks to ensure that the same R isn't reused.
Problems can also arise if K is compromised. For example, an attacker who compromises K can perform a man-in-the-middle attack and listen to all cleartext communication. Such an attacker could send messages between the two parties while pretending to be both the legitimate SIM card operator and the SIM card. The greater risk is that an attacker can record communications and any messages exchanged during the key agreement, and later decrypt those communications by using the captured R values. An attacker could then determine the past session keys and use them to decrypt the recorded traffic.

Attack Models for Key Agreement Protocols

There are different notions of security in key agreement protocols as well as three main attack models that depend on the information the protocol leaks. From weakest to strongest, these are **the eavesdropper**

Attack Models for Key Agreement Protocols

There are different notions of security in key agreement protocols as well as three main attack models that depend on the information the protocol leaks. From weakest to strongest, these are **the eavesdropper**, **the data leak**

Attack Models for Key Agreement Protocols

There are different notions of security in key agreement protocols as well as three main attack models that depend on the information the protocol leaks. From weakest to strongest, these are **the eavesdropper**, **the data leak**, and **the breach**

Attack Models for Key Agreement Protocols

There are different notions of security in key agreement protocols as well as three main attack models that depend on the information the protocol leaks. From weakest to strongest, these are **the eavesdropper**, **the data leak**, and **the breach**:

The eavesdropper This attacker observes the messages exchanged between the two legitimate parties running a key agreement protocol and can record, modify, drop, or inject messages. To protect against an eavesdropper, a key agreement protocol must not leak any information on the shared secret established.

The data leak In this model, the attacker acquires the session key and all temporary secrets (such as SK in the telecom protocol example) from one or more executions of the protocol, but not the long-term secrets (like K in that same protocol).

The data leak In this model, the attacker acquires the session key and all temporary secrets (such as SK in the telecom protocol example) from one or more executions of the protocol, but not the long-term secrets (like K in that same protocol).

The breach (or corruption) In this model, the attacker learns the long-term key of one or more of the parties. Once a breach occurs, security is no longer attainable because the attacker can impersonate one or both parties in subsequent sessions of the protocol. Nonetheless, the attacker shouldn't be able to recover secrets from sessions executed before gathering the key.

Security goals

A key agreement protocol can be designed to satisfy several security goals. The four most relevant ones are described here, in order from simplest to most sophisticated.

Security goals

A key agreement protocol can be designed to satisfy several security goals. The four most relevant ones are described here, in order from simplest to most sophisticated.

Authentication Each party should be able to authenticate the other party. That is, the protocol should allow for mutual authentication. Authenticated key agreement (AKA) occurs when a protocol authenticates both parties.

Security goals

A key agreement protocol can be designed to satisfy several security goals. The four most relevant ones are described here, in order from simplest to most sophisticated.

Authentication Each party should be able to authenticate the other party. That is, the protocol should allow for mutual authentication. Authenticated key agreement (*AKA*) occurs when a protocol authenticates both parties.

Key control Neither party should be able to choose the final shared secret or coerce it to be in a specific subset. The previous telecom protocol lacks this property.

Forward secrecy This is the assurance that even if all long-term secrets are exposed, shared secrets from previous executions of the protocol won't be able to be computed, even if an attacker records all previous executions or is able to inject or modify messages from previous executions. A forward-secret protocol guarantees that even if you have to deliver your devices and their secrets to some authority or other, they won't be able to decrypt your prior encrypted communications.

Forward secrecy This is the assurance that even if all long-term secrets are exposed, shared secrets from previous executions of the protocol won't be able to be computed, even if an attacker records all previous executions or is able to inject or modify messages from previous executions. A forward-secret protocol guarantees that even if you have to deliver your devices and their secrets to some authority or other, they won't be able to decrypt your prior encrypted communications.

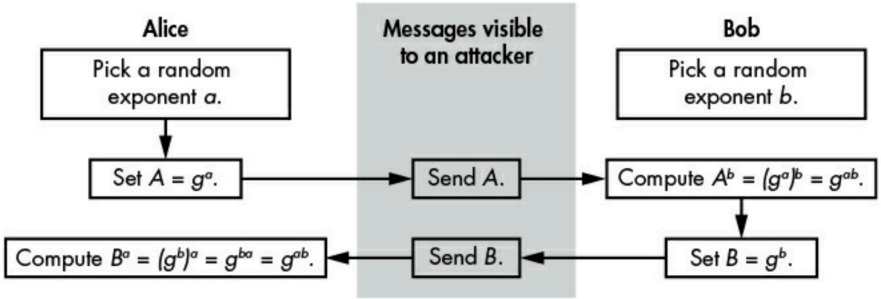
Resistance to key-compromise impersonation (KCI) KCI occurs when an attacker compromises a party's long-term key and is able to use it to impersonate another party. The 3G/4G key agreement protocol allows trivial key-compromise impersonation because both parties share the same key K. A key agreement protocol should ideally prevent this kind of attack.

Anonymous Diffie–Hellman

Anonymous Diffie–Hellman is the simplest of the Diffie–Hellman protocols. The participants have no identity that can be verified by either party, and neither party holds a long-term key.

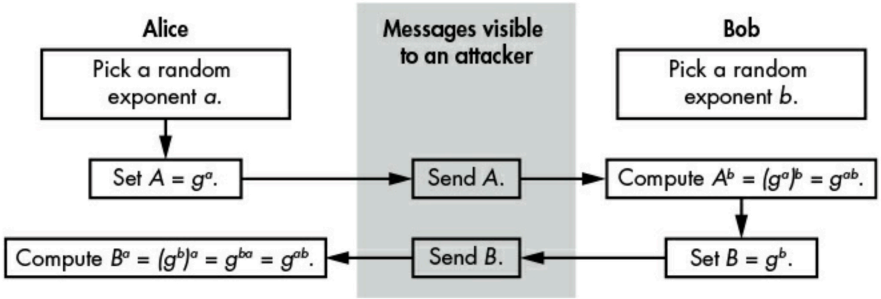
Anonymous Diffie–Hellman

Anonymous Diffie–Hellman is the simplest of the Diffie–Hellman protocols. The participants have no identity that can be verified by either party, and neither party holds a long-term key.

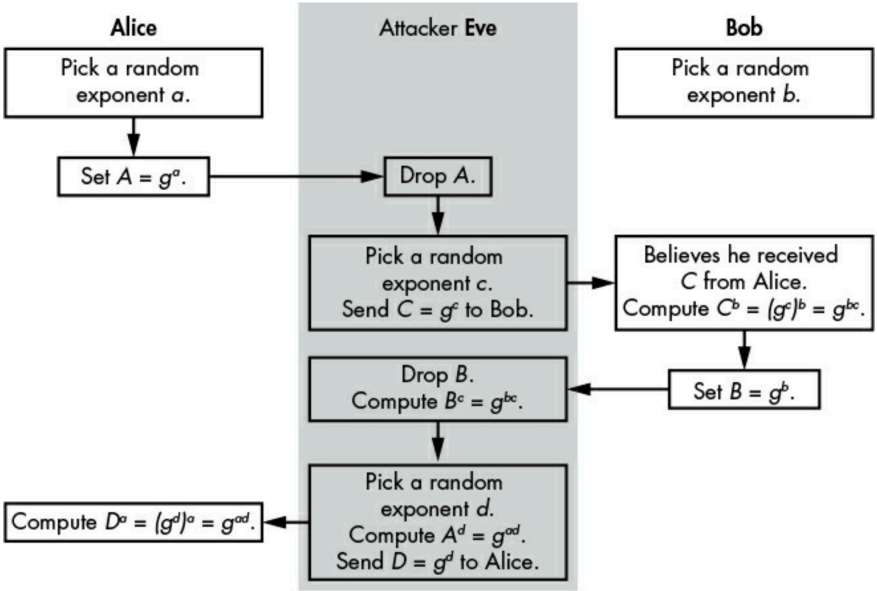


Anonymous Diffie–Hellman

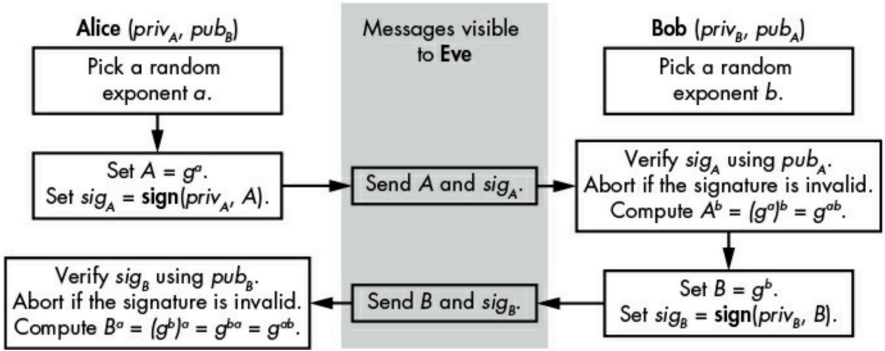
Anonymous Diffie–Hellman is the simplest of the Diffie–Hellman protocols. The participants have no identity that can be verified by either party, and neither party holds a long-term key.



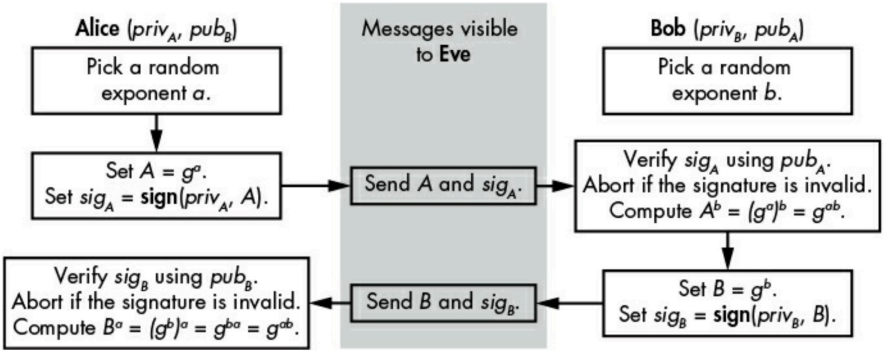
Pure, simple, but only secure against the laziest of attackers. Anonymous DH can be taken down with a man-in-the-middle attack. An eavesdropper simply needs to intercept messages and pretend to be Bob (to Alice) and pretend to be Alice (to Bob).


Alice
Attacker Eve
Bob
Pick a random exponent a.
Pick a random exponent b.
Set A = g^a.
Drop A.
Pick a random exponent c. Send C = g^c to Bob.
Believes he received C from Alice. Compute C^b = (g^c)^b = g^bc.
Drop B. Compute B^c = g^bc.
Set B = g^b.
Pick a random exponent d. Compute A^d = g^ad. Send D = g^d to Alice.
Compute D^a = (g^d)^a = g^ad.

Authenticated Diffie–Hellman



Authenticated Diffie–Hellman



Authenticated DH equips the two parties with both a private and a public key, thereby allowing Alice and Bob to sign their messages in order to stop Eve from sending messages on their behalf.

Security Against Eavesdroppers

- *Authenticated DH* is secure against eavesdroppers because attackers can't learn any bit of information on the shared secret gab since they ignore the *DH* exponents.

Security Against Eavesdroppers

- *Authenticated DH* is secure against eavesdroppers because attackers can't learn any bit of information on the shared secret gab since they ignore the *DH* exponents.
- It also provides forward secrecy: even if an attacker corrupts any of the parties at some point, as in the breach attack model discussed earlier, they would learn the private signing keys but not any of the ephemeral DH exponents; hence, they'd be unable to learn the value of any previously shared secrets.

Security Against Eavesdroppers

- *Authenticated DH* is secure against eavesdroppers because attackers can't learn any bit of information on the shared secret gab since they ignore the *DH* exponents.
- It also provides forward secrecy: even if an attacker corrupts any of the parties at some point, as in the breach attack model discussed earlier, they would learn the private signing keys but not any of the ephemeral DH exponents; hence, they'd be unable to learn the value of any previously shared secrets.
- *Authenticated DH* also prevents any party from controlling the value of the shared secret. Alice can't craft a special value of a in order to predict the value of g^{ab} because she doesn't control b, which influences gab as much as a does. ($a = 0$ should be refused by the protocol).

Authenticated DH isn't secure against all types of attack. For one thing, Eve can record previous values of A and sig_A and replay them later to Bob, in order to pretend to be Alice. Bob will then believe that he's sharing a secret with Alice when he isn't, even though Eve would not be able to learn that secret.

Security Against Data Leaks

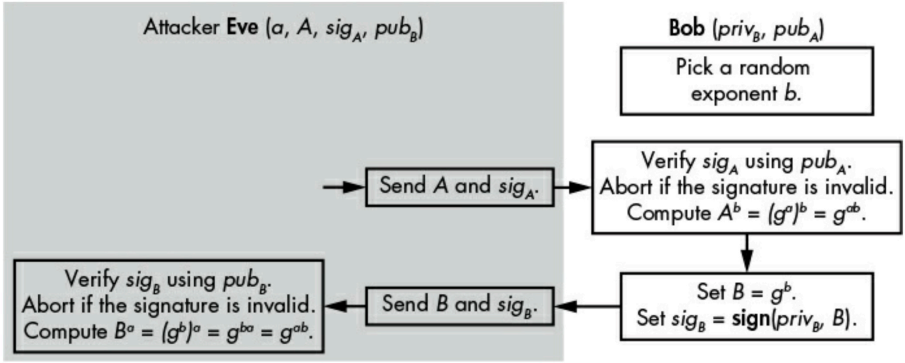
Authenticated DH's vulnerability to data leak attackers is of greater concern.

Security Against Data Leaks

Authenticated DH's vulnerability to data leak attackers is of greater concern.In this type of attack, the attacker learns the value of ephemeral, short-term secrets (namely, the exponents a and b) and uses that information to impersonate one of the communicating parties.

Security Against Data Leaks

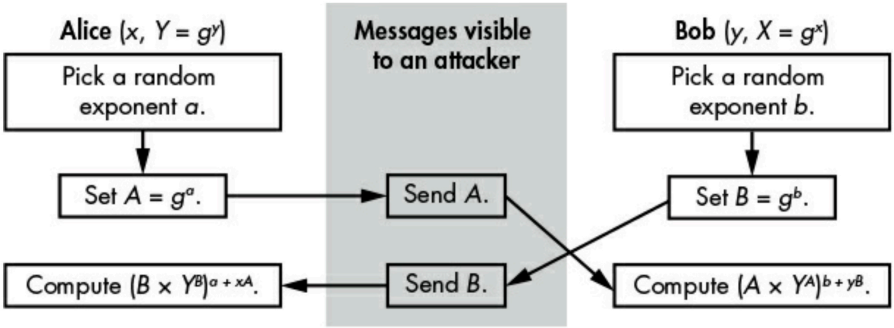
Authenticated DH's vulnerability to data leak attackers is of greater concern.In this type of attack, the attacker learns the value of ephemeral, short-term secrets (namely, the exponents *a* and *b*) and uses that information to impersonate one of the communicating parties.If Eve is able to learn the value of an exponent a along with the matching values of A and sigA sent to Bob, she could initiate a new execution of the protocol and impersonate Alice.

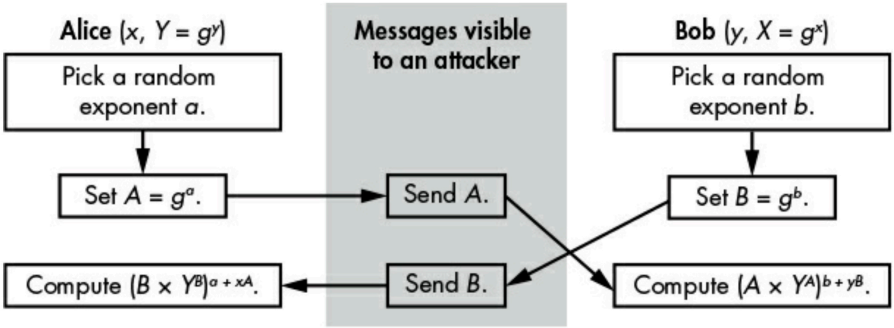


One way to make authenticated DH secure against the leak of ephemeral secrets is to integrate the long-term keys into the shared secret computation so that the shared secret can't be determined without knowing the long-term secret.

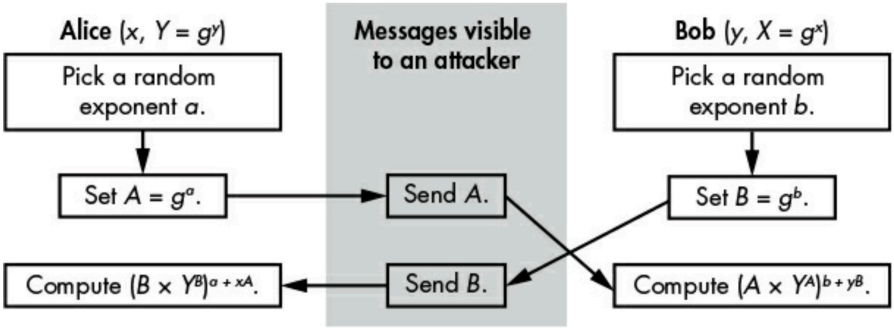
Menezes–Qu–Vanstone (*MQV*)

MQV is Diffie–Hellman on steroids. It's more secure than *authenticated DH*, and it improves on authenticated *DH*'s performance properties. In particular, *MQV* allows users to send only two messages, independently of each other, in arbitrary order. Other benefits are that users can send shorter messages than they would be able to with authenticated DH, and they don't need to send explicit signature or verification messages. In other words, you don't need to use a signature scheme in addition to the Diffie–Hellman function.


Alice (x, $Y = g^y$)
Pick a random exponent a.
Set $A = g^a$.
Compute $(B \times Y^B)^{a + xA}$.
Messages visible to an attacker
Send A.
Send B.
Bob (y, $X = g^x$)
Pick a random exponent b.
Set $B = g^b$.
Compute $(A \times Y^A)^{b + yB}$.



The x and y are Alice and Bob's respective long-term private keys, and X and Y are their public keys.



The x and y are Alice and Bob's respective long-term private keys, and X and Y are their public keys.Bob and Alice start out with their own private keys and each other's public keys, which are g to the power of a private key.

$$\begin{aligned}\left(BY^B\right)^{a+xA} &= \left(g^b g^{yB}\right)^{a+xA} = g^{(b+yB)(a+yA)} \\ \left(AY^A\right)^{b+yB} &= \left(g^a g^{xA}\right)^{b+yB} = g^{(b+yB)(a+yA)}.\end{aligned}$$

$$\begin{aligned}\left(BY^B\right)^{a+xA} &= \left(g^b g^{yB}\right)^{a+xA} = g^{(b+yB)(a+yA)} \\ \left(AY^A\right)^{b+yB} &= \left(g^a g^{xA}\right)^{b+yB} = g^{(b+yB)(a+yA)}.\end{aligned}$$

Thus, there is a shared secret!

$$\begin{aligned}\left(BY^B\right)^{a+xA} &= \left(g^b g^{yB}\right)^{a+xA} = g^{(b+yB)(a+yA)} \\ \left(AY^A\right)^{b+yB} &= \left(g^a g^{xA}\right)^{b+yB} = g^{(b+yB)(a+yA)}.\end{aligned}$$

Thus, there is a shared secret!

Unlike *authenticated DH*, *MQV* can't be broken by a mere leak of the ephemeral secrets. Knowledge of a or b won't let an attacker determine the final shared secret because they would need the long-term private keys to compute it.

$$\begin{aligned}\left(BY^B\right)^{a+xA} &= \left(g^b g^{yB}\right)^{a+xA} = g^{(b+yB)(a+yA)} \\ \left(AY^A\right)^{b+yB} &= \left(g^a g^{xA}\right)^{b+yB} = g^{(b+yB)(a+yA)}.\end{aligned}$$

Thus, there is a shared secret!

Unlike *authenticated DH*, *MQV* can't be broken by a mere leak of the ephemeral secrets. Knowledge of a or b won't let an attacker determine the final shared secret because they would need the long-term private keys to compute it.

What happens in the strongest attack model, the breach model, where a long-term key is compromised? If Eve compromises Alice's long-term private key x, the previously established shared secrets are safe because their computation also involved Alice's ephemeral private keys.

However, MQV doesn't provide perfect forward secrecy because of the following attack.

However, MQV doesn't provide perfect forward secrecy because of the following attack.If Eve intercepts Alice's A message and replaces it with her $A = g^a$ for some a that Eve has chosen. In the meantime, Bob sends B to Alice and computes the shared key. If Eve later compromises Alice's long-term private key x, she can determine the key that Bob had computed during this session. This breaks forward secrecy, since Eve has now recovered the shared secret of a previous execution of the protocol.

However, MQV doesn't provide perfect forward secrecy because of the following attack.If Eve intercepts Alice's A message and replaces it with her $A = g^a$ for some a that Eve has chosen. In the meantime, Bob sends B to Alice and computes the shared key. If Eve later compromises Alice's long-term private key x, she can determine the key that Bob had computed during this session. This breaks forward secrecy, since Eve has now recovered the shared secret of a previous execution of the protocol.In practice the risk can be eliminated by a key-confirmation step that would have Alice and Bob realize that they don't share the same key, and they would therefore abort the protocol before deriving any session keys.

What not to do!

Not Hashing the Shared Secret

The creation of a shared secret, g^{ab}, that concludes a DH session exchange is not a key itself.

Not Hashing the Shared Secret

The creation of a shared secret, g^{ab}, that concludes a DH session exchange is not a key itself. A symmetric key should look random, and each bit should either be 0 or 1 with the same probability. But $g^a b$ is not a random string; it is a random element within some mathematical group whose bits may be biased toward 0 or 1.

Not Hashing the Shared Secret

The creation of a shared secret, g^{ab}, that concludes a DH session exchange is not a key itself. A symmetric key should look random, and each bit should either be 0 or 1 with the same probability. But $g^a b$ is not a random string; it is a random element within some mathematical group whose bits may be biased toward 0 or 1.

Suppose that we are using $\mathbb{Z}_{13}^*$ and $g = 2$. If g's exponent is random, you'll get a random element of $\mathbb{Z}_{13}^*$, but the encoding of a $\mathbb{Z}_{13}^*$ element as a 4-bit string won't be uniformly random: not all bits will have the same probability of being a 0 or a 1.

Not Hashing the Shared Secret

The creation of a shared secret, g^{ab}, that concludes a DH session exchange is not a key itself. A symmetric key should look random, and each bit should either be 0 or 1 with the same probability. But $g^a b$ is not a random string; it is a random element within some mathematical group whose bits may be biased toward 0 or 1.

Suppose that we are using $\mathbb{Z}_{13}^*$ and $g = 2$. If g's exponent is random, you'll get a random element of $\mathbb{Z}_{13}^*$, but the encoding of a $\mathbb{Z}_{13}^*$ element as a 4-bit string won't be uniformly random: not all bits will have the same probability of being a 0 or a 1.In $\mathbb{Z}_{13}^*$, seven values have 0 as their most significant bit, but only five have 1 as their most significant bit. That is, this bit is 0 with probability $\frac{7}{12}$, whereas, ideally, a random bit should be 0 with probability $\frac{1}{2}$. Moreover, the 4-bit sequences 1101, 1110, and 1111 will never appear.

Legacy Diffie–Hellman in *TLS*

The *TLS* protocol is the security behind *HTTPS* secure websites as well as the secure mail transfer protocol (*SMTP*). *TLS* takes several parameters, including the type of Diffie–Hellman protocol it will use, though most *TLS* implementations still support *anonymous DH* for legacy reasons, despite its insecurity.

Unsafe Group Parameters

In January 2016, the maintainers of the OpenSSL toolkit fixed a high- severity vulnerability (CVE-2016-0701) that allowed an attacker to exploit unsafe Diffie–Hellman parameters. The root cause of the vulnerability was that OpenSSL allowed users to work with unsafe DH group parameters (namely, an unsafe prime p) instead of throwing an error and aborting the protocol altogether before performing any arithmetic operation.


TO BE CONTINUED...