

Cryptographic Engineering

working draft

Matthew D. Green
Johns Hopkins University

Excerpt: Chapter Chapter 7, Public-key cryptography

§7.1 Introduction; §7.1.1 Key exchange (book pages 1-4)

Posted for 601.445/601.645 Practical Cryptographic Systems, Fall 2026
Lecture 7, Wed Sep 23: public-key cryptography I (groups, Diffie–Hellman)

<https://practicalcrypto.org>

This is an excerpt from an unfinished book draft: expect gaps, rough edges, and errors, and note that section numbers and page numbers may change in later drafts. Cross-references to other chapters point to material that is not included here. Copyright © 2026 Matthew D. Green. All rights reserved. Posted for the students of the course above; **please do not redistribute** — link to the course site instead. Built 2026-09-06.

Chapter 7

Public-key cryptography

So far we have restricted ourselves to a discussion of symmetric-key cryptography, where all parties in the communication share a copy of the same key. With a few exceptions, we have mostly ignored the problem of how those parties obtained their keys. These exceptions themselves are informative: for example, protocols such as Needham-Schroeder and Kerberos allow us to leverage one shared key relationship into relationships with many different parties, but require us to use online trusted parties for every new communication request. Clearly we do not use such parties to communicate securely with websites today. So how do we establish communications over the Internet?

The answer to this question is *public key cryptography*. Public key schemes allow participants to communicate securely over an insecure channel, even if the communicating parties do not previously share each others' secrets.

7.1 Introduction to public-key cryptography

As computers became more powerful in the 1970s, a number of researchers began investigating new ways to use sophisticated mathematical techniques in cipher design. It was during this era that cryptographers in both the classified agencies and the public research community hit on the same question: *is it possible to develop techniques that allow two parties to exchange keys over a public channel?*

This is a surprisingly difficult problem to solve. Using symmetric cryptographic techniques, the only way to exchange keys is to do so in secret, meaning that keys must be carefully exchanged through a channel that the attacker cannot see. This new idea was revolutionary: here parties could somehow communicate right out in the open, for example over a broadcast radio channel that the adversary could easily listen in on. This would make key exchange enormously easier, but it also wasn't obvious if such a thing could even be done. Certainly developing it would require entirely new techniques.

The first researchers to make a public breakthrough were Whitfield Diffie and Martin Hellman at Stanford University. Diffie and Hellman had been studying the idea of using

number theory problems as the basis for cryptography, including one specific problem known as the *discrete logarithm* problem..

At a superficial level, the discrete logarithm is similar to the classical logarithm: given some value $A = a^b$ where the base a is known, that problem is to find the value of the exponent b . When exponentiation is computed over integers or floating point numbers, logarithms are relatively easy for computers to solve. In fact, many computer processors include fast instructions for solving them.

However, Diffie and Hellman knew that exponentiation could also be realized in mathematical structures called *cyclic algebraic groups*. While these groups can be built from many different mathematical components, a simple instantiation uses modular arithmetic over the integers (quite similar to the arithmetic in describing the Vigenère cipher.) Let p be some large prime number, which does not need to be secret. Given a specific integer g that is called a *generator*, the modular exponentiation function $A = g^a \bmod p$ is relatively efficient to compute for some integer a .¹ The discrete logarithm problem is to *invert* this function: in other words, to find a given knowledge of A, g and p . To the best of Diffie and Hellman’s knowledge (and indeed, our knowledge today) it is hard for even enormously powerful computers to solve this problem when the parameters are selected appropriately.

7.1.1 Key Exchange

What Diffie-Hellman had, at this point, was something cryptographers call a *one-way function*. This is a function where it is easy to compute in one direction (given a , compute $A \leftarrow g^a \bmod p$) but where it is enormously time consuming to go backwards, *i.e.*, from the output A to the initial value a . While cryptographers now know that one-way functions are a precondition to public-key encryption, we also know that they’re not sufficient. Diffie and Hellman still required a way to use their one-way function so that two parties could securely exchange keys.

To do this, the pair realized that they could obtain a key exchange protocol if they added one more idea. Their solution relies on the fact that modular exponentiation is *commutative*: specifically, given two integers a and b , it does not matter whether one computes $(g^b)^a \bmod p$ or $(g^a)^b \bmod p$: the two results are exactly the same!

The Diffie-Hellman protocol relies on this mathematical identity. Two participants, who we’ll call Alice and Bob, first agree on some non-secret parameters g and p . They each select a secret integer at random: we’ll call those a and b respectively. Their goal is to compute a shared secret S using modular exponentiation, of the form $S = g^{ab} \bmod p$. Alice begins by choosing a and sending the following “key share” to Bob:

$$\text{Alice} \rightarrow \text{Bob} : A = g^a \bmod p$$

¹Note that exponentiation can be decomposed into a much smaller number of repeated operations. An example: if $a = 16$ then g^{16} would naïvely require 16 modular multiplications. However, by re-writing this as $((((g^2)^2)^2)^2) \bmod p$ it can be computed in only $\log_2(16) = 4$ multiplications. This general rule can be extended to any a , using an algorithm called “square and multiply” (see §??).

Next Bob does the same, this time using his secret b :

$$\text{Alice} \leftarrow \text{Bob} : B = g^b \bmod p$$

Finally, Alice and Bob each combine the received value with their own secret. Alice computes $S = B^a \bmod p$. And Bob, in turn, computes $S = A^b \bmod p$. When fully worked out, these values are the same, since $(g^b)^a \equiv (g^a)^b \bmod p$. Yet critically, an attacker who observes the parties' communications will learn only A and B , from which they cannot easily compute S . With a shared secret in hand, Alice and Bob can now use this value to derive an encryption key for a secure symmetric encryption scheme.

Security of Diffie-Hellman. The Diffie-Hellman protocol was a remarkable invention, since it allowed two parties to establish a key over a *completely public channel* without an adversary learning their shared key. It was also one of the first cryptographic schemes to base its security on a well-studied mathematical problem. Of course, this last claim is a slight exaggeration: in fact, while Diffie-Hellman clearly depends on the discrete logarithm problem being hard, it is not known whether the scheme and problem are equivalent; instead, the actual security of the Diffie-Hellman protocol can be reduced to solving a related mathematical problem called the Diffie-Hellman problem.² (This would not be the last time that a cryptographers invented a cryptosystem, then justified its security using a new mathematical problem that was, essentially, the cryptosystem itself.) Nonetheless, to this day the best known approach to breaking Diffie-Hellman is to solve discrete logarithms.

Although the original version of Diffie-Hellman described above uses cyclic groups based on finite fields (modular integer arithmetic), Diffie-Hellman can also be realized in other cyclic groups. The most popular alternative setting uses a type of cyclic group based on points that satisfy a specific *elliptic curve* equation, typically over one of a small number of standardized elliptic curves (see §7.3.8.) Here, all you need to know for the most basic understanding is that these groups also feature a generator, usually denoted as P , and use *point addition* rather than modular multiplication for their group operation. Hence, we will often notate elliptic-curve Diffie-Hellman (ECDH) key exchanges using values such as aP, bP and abP , where each value is an elliptic curve point (see Figure 7.1.) The reason that practitioners prefer the elliptic curve variant is that it blocks the use of some modern cryptanalytic algorithms such as the *index calculus* methods that can dramatically speed up the process of solving discrete logarithms in finite fields.³ The result of this is that elliptic

²This problem, stated roughly, is to solve the Diffie-Hellman key exchange given only the information that would be available to an eavesdropper: namely, given $(g, p, A = g^a \bmod p, B = g^b \bmod p)$ for random a, b , find g^{ab} . Clearly an algorithm for solving the discrete logarithm would make this problem easy, by allowing an attacker to recover a or b . It is not known if the inverse is true: that is, whether an algorithm for solving the Diffie-Hellman problem would provide a way to solve the discrete logarithm problem. Hence it is at least theoretically possible for the discrete logarithm to remain hard, but for Diffie-Hellman itself to somehow be insecure.

³While index calculus methods do not make finite-field groups insecure, it requires that we use larger

curve Diffie-Hellman can use smaller key shares, which makes this ECDH variant preferable for use in TLS and many other security protocols.

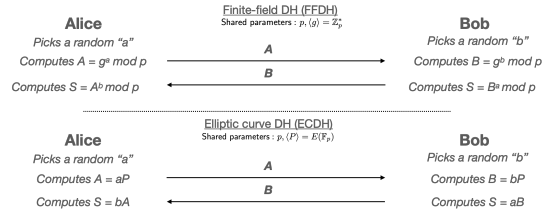


Figure 7.1: The finite-field (top) and elliptic-curve Diffie-Hellman (bottom) protocols. In both protocols p is some prime number.

It is also worth mentioning that recent advances in quantum computing pose a threat to the hardness of the discrete logarithm problem and hence Diffie-Hellman, in both its finite-field and elliptic curve variants. This is due to the development of efficient quantum algorithms that can solve discrete logarithms. Fortunately, we do not yet know how to build quantum computers that are powerful enough to run these algorithms against real Diffie-Hellman implementations. Nonetheless, the fear of future quantum computers has motivated the development of replacements for Diffie-Hellman and other vulnerable protocols. We'll talk about this quite a bit more in §??.

Meddlers in the middle. While Diffie-Hellman works well against an attacker who simply listens to communications (called a *passive attacker* or *eavesdropper*), it can be vulnerable to a more powerful attacker who can change or modify messages sent over the channel. Against this attacker, the problem with the basic Diffie-Hellman protocol is that, in its basic form, it lacks *authentication*. In practice, it is relatively easy for an active attacker to impersonate the two parties to each other.

For example, imagine that an attacker Mallory sits on the channel between Alice and Bob and impersonates the participants: for example, Mallory pretends to be Bob when talking to Alice, and pretends to be Alice when talking to Bob (this is particularly easy to do on wireless connections, or when Mallory controls a network router.) If Alice attempts to run the Diffie-Hellman protocol with Bob, she will actually be establishing a shared key with Mallory! If Mallory is clever, they may be able to perform the same trick against Bob — causing Bob to establish his own key with Mallory. Yet they will instead be encrypting messages to Mallory. Mallory can then decrypt each message sent by one party, read it, and then re-encrypt under the key it shares with the other party. This is known as an

values for p , on the order of 2048 and 3,072 bits rather than the 256 bits that is possible in elliptic curve variants.