Fundamentals of Modern Cryptography

BRUCE MOMJIAN

This presentation explains the fundamentals of modern cryptographic methods.

https://momjian.us/presentations

Last updated: April 2023

What Cryptography Can Accomplish

Confidentiality Only the other party can read the original messages

Authenticity Verify who is on the other end of the communication channel

Integrity No other party can change or add messages

Other features are often desired, e.g., non-repudiation.

Outline

1. Ciphers and hashes
2. Big numbers
3. Primes
4. Private communication
5. Public/private-key communication

1 Ciphers and Hashes: Ciphers

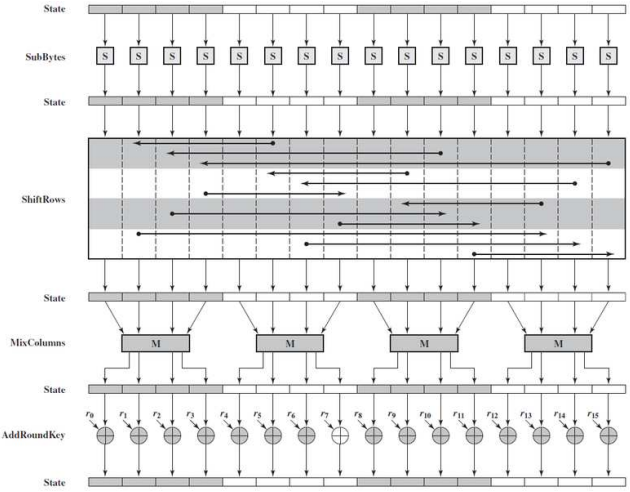
$$
\begin{aligned}
encrypt(message, secret) &= cipher \\
decrypt(cipher, secret) &= message \\
decrypt(encrypt(message, secret), secret) &= message
\end{aligned}
$$

Ciphers map a message to a same-length unique ciphertext, which can be reversed.

AES (Advanced Encryption Standard)

- 128-bit block cipher
- Supports key lengths of 128, 192, and 256 bits, e.g., AES128 uses a 128-bit key
- Uses confusion (substitution) and diffusion; see https://en.wikipedia.org/wiki/Confusion_and_diffusion
- No known attack except for exhaustive key search (this is ideal)
- AES-specific CPU instructions speed processing 3–10x, e.g., AES-NI (check */proc/cpuinfo* for "aes"), see https://www.cyberciti.biz/faq/how-to-find-out-aes-ni-advanced-encryption-enabled-on-linux-system/
- Elegant algorithm based on polynomials over finite fields, see https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
- Trusted by the US government for top secret communication

One Cycle of AES



AES128 requires 10 cycles.

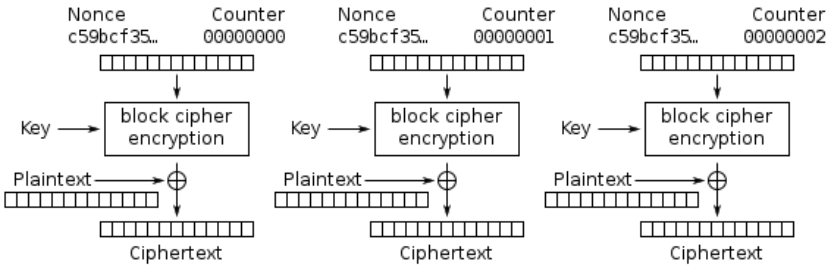
https://commons.wikimedia.org/wiki/File:AES_Encryption_Round.png

Cipher Modes

$$
\begin{aligned}
message &= 128bit_1 || 128bit_2 || \ldots \\
encrypt(message, secret) &= encrypt(128bit_1, secret) || encrypt(128bit_2, secret) || \ldots \\
block\ encrypt(128bit_1, secret) &= cipher(secret, 128bit_1\ XOR\ (previous\ output\ OR\ nonce)) \quad (CBC) \\
stream\ encrypt(128bit_1, secret) &= 128bit_1\ XOR\ cipher(secret, counter||nonce) \quad (CTR, GCM)
\end{aligned}
$$

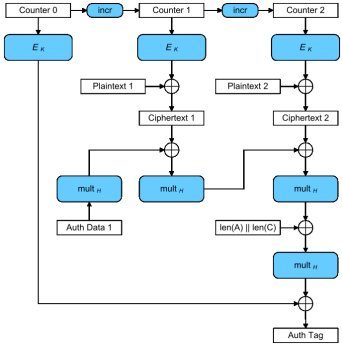
Stream encryption built on block ciphers is particularly useful for streaming protocols, e.g., SSH. You can generate a 128-bit encryption block and XOR the 16 bytes individually.

CTR (Counter) Illustrated



https://commons.wikimedia.org/wiki/File:CBC_encryption.svg

GCM (Galois Counter Mode) Illustrated



GCM is similar to CTR mode but also creates an integrity tag that can be used to verify that the entire message was created with the same secret key and unmodified.

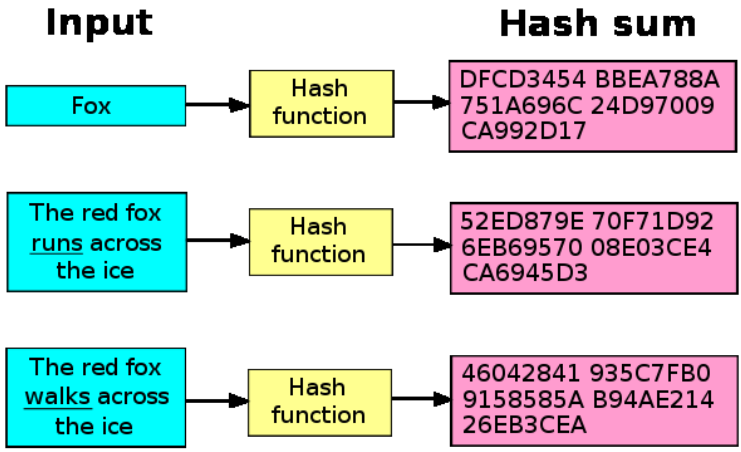
https://en.wikipedia.org/wiki/Galois/Counter_Mode\#/media/File:GCM-Galois_Counter_Mode.svg

Hash

$$\begin{aligned} hash(message) &= hash \\ function(hash) &\neq message \end{aligned}$$

A hash maps a variable-length value to a fixed-length value, with minimal collisions. Collisions are likely at $\sqrt{hash\ space}$, e.g., a hash algorithm with 2^{256}possible outputs is likely to generate a collision among 2^{128} outputs. The hash name indicates the hash output bit length, e.g., SHA-256, SHA-384.

Hashing Illustrated



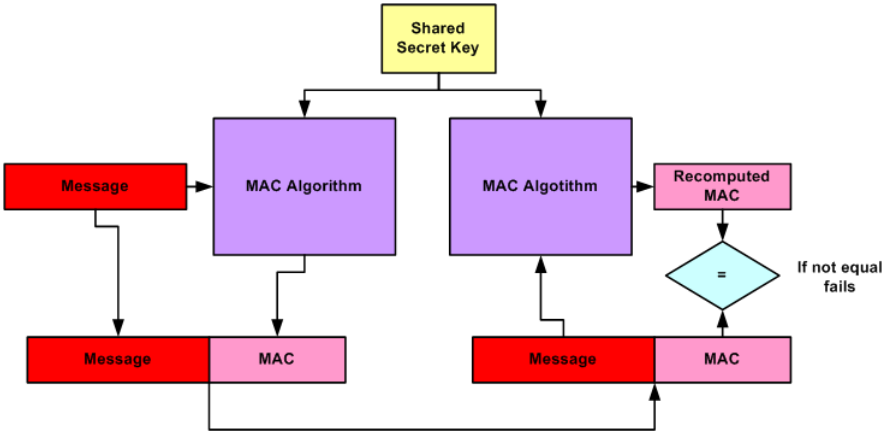
https://commons.wikimedia.org/wiki/File:Hash_function_long.svg

MAC (Message Authentication Code)

$$\begin{aligned} hash(message, secret) &= hash \\ hash(message, no\ secret) &\neq matching\ hash \end{aligned}$$

- Hashing (or limited block encryption) allows you to safely prove you know a secret
- Someone who also knows the secret can compute the same MAC and check that it matches
- Others who see the MAC can't reverse it to divulge the secret
- A message authentication code (MAC) proves that the message was created by someone who knows the secret
- Multiple pieces of information can be combined in a single hash, e.g.,
  - often the message, message length, and secret key are combined to produce a single hash value
  - this proves the message was produced by someone who knows the secret

MAC Illustrated



https://commons.wikimedia.org/wiki/File:Cryptographic_MAC_based_message_authentication.png

Split Keys

While it is possible to merge secrets in a single hash, it is also possible to split a secret into parts so all parts are needed to reconstruct the secret, e.g two parts:

$$
\begin{aligned}
key1 &= \textit{random number} \\
\textit{master secret XOR } secret1 &= secret2 \\
\textit{to reconstruct} & \\
secret1 \textit{ XOR } secret2 &= \textit{master secret}
\end{aligned}
$$

This can be repeated to split a key into any number of parts; see https://en.wikipedia.org/wiki/Secret_sharing.

2 Big Numbers: Exponentiation

$$a^b = a \times a \times a ... (b \text{ times})$$

e.g.,

$$2^5 = 2 \times 2 \times 2 \times 2 \times 2$$

$$7\text{E}9 = 7 \times 10^9 = 7,000,000,000$$

Logarithms

$$\begin{aligned} a^b &= c \\ b \times \log a &= \log c \\ b &= \frac{\log c}{\log a} \end{aligned}$$

CPU Register Sizes

$$\begin{aligned} \text{32 bit } 2^{32} &\cong 4\text{E}9 \\ \text{64 bit } 2^{64} &\cong 2\text{E}19 \end{aligned}$$

Real World Sizes

- Seconds in a billion years: 3E16
- Atoms in the universe: 1E80
- Possible chess games: 1E120

Cryptography Sizes

- AES128 (128 bit): 3E38
- ECDHE prime (224 bit): 2E67
- AES256, SHA256 (256 bit): 1E77
- RSA prime (1024 bit): 2E308
- DH prime, RSA composite (2048 bit): 4E616
  - strength equivalent to a 112-bit symmetric cipher (3E35)
- Random values with range 2^b likely repeat after $2^{b/2}$values
- Adding a bit doubles the strength, adding two bits quadruples the strength
- Doubling the number of bits exponentially increases the strength

This video explains the difficulty of breaking large computed secrets, https://www.youtube.com/watch?v=S9JGmA5_unY.

https://en.wikipedia.org/wiki/Key_size

3 Mathematical Concepts

1. Exponential commutativity
2. Prime number
3. Modulus
4. Finite field
5. Generator
6. Primitive element
7. Trapdoor function

Exponentiation Commutativity

$$g^{x^y} = g^{x \cdot y} = g^{y \cdot x} = g^{y^x}$$

Prime Number

A prime number is a natural number greater than one that is not a product of two smaller natural numbers; see https://en.wikipedia.org/wiki/Prime_number.

Modulus

The modulus is the remainder of division after one number is divided by another; see https://en.wikipedia.org/wiki/Modulo_operation:

$$\begin{aligned} 9 \bmod 7 &= 2 \\ 9 \,\%\, 7 &= 2 \end{aligned}$$

Finite Field

A finite/Galois field is a field that contains a finite number of elements and where multiplication, addition, subtraction and division are defined and satisfy certain basic rules; see https://en.wikipedia.org/wiki/Finite_field.

Generator

Six raised to an integer power modulus 7 *generates* a finite field containing 1 and 6:

$$
\begin{aligned}
6^0 \bmod 7 &= 1 \\
6^1 \bmod 7 &= 6 \\
6^2 \bmod 7 &= 1 \\
6^3 \bmod 7 &= 6
\end{aligned}
$$

Primitive Element

Three is a *primitive element* of 7 because it generates the full set of the finite field values less than 7:

$$
\begin{aligned}
3^1 \bmod 7 &= 3 \\
3^2 \bmod 7 &= 2 \\
3^3 \bmod 7 &= 6 \\
3^4 \bmod 7 &= 4 \\
3^5 \bmod 7 &= 5 \\
3^6 \bmod 7 &= 1 \\
3^7 \bmod 7 &= 3 \\
3^8 \bmod 7 &= 2
\end{aligned}
$$

Generating all numbers of a finite field in unpredictable order is fundamental to cryptography.

Trapdoor Function

Trap door functions are functions where it is difficult to compute previously-generated values; see https://en.wikipedia.org/wiki/Trapdoor_function. Here is the previous function in numeric output order, which is a good example of a trap door function:

$$
\begin{aligned}
1 &= 3^{6n} \bmod 7 \\
2 &= 3^{6n+2} \bmod 7 \\
3 &= 3^{6n+1} \bmod 7 \\
4 &= 3^{6n+4} \bmod 7 \\
5 &= 3^{6n+5} \bmod 7 \\
6 &= 3^{6n+3} \bmod 7
\end{aligned}
$$

for any positive integer n. You can see it is difficult to determine the previously-generated values.

4 Private Communication: Ephemeral Diffie–Hellman (DHE)

Assume g is a (sub-group) generator of (safe) prime p:

Alice	Bob
agree on g and p	agree on g and p
generate random x	generate random y
compute $g^x \bmod p$	compute $g^y \bmod p$
send $g^x \bmod p \rightarrow$	$\leftarrow$ send $g^y \bmod p$
receive $g^y \bmod p \leftarrow$	$\rightarrow$ receive $g^x \bmod p$
use x, compute $(g^y \bmod p)^x \bmod p$	use y, compute $(g^x \bmod p)^y \bmod p$
use as shared key $g^{y^x} \bmod p$	use as shared key$g^{x^y} \bmod p$

Viewing transmission of g, p, $g^x \bmod p$, and $g^y \bmod p$ does not allow easy discovery of x, y, *or* $g^{x^y} \bmod p$ (the secret key). This seems trivial until you realize it is being done using 2048-bit (3E616) values. Exponentiation by high powers can be performed efficiently using exponentiation by squaring (binary exponentiation), optimized via modular exponentiation.

The Result

Both users have the same secret key:

$$g^{y^x} \bmod p \quad = \quad g^{x^y} \bmod p$$

while anyone viewing the exchange cannot derive the secret key. There is no information in the key. Instead, the computed key is used to generate a session key for encryption.

Fake Addition Key Exchange Example

Pretend you can't reverse the modulo computation (this is the effect of a large exponent):

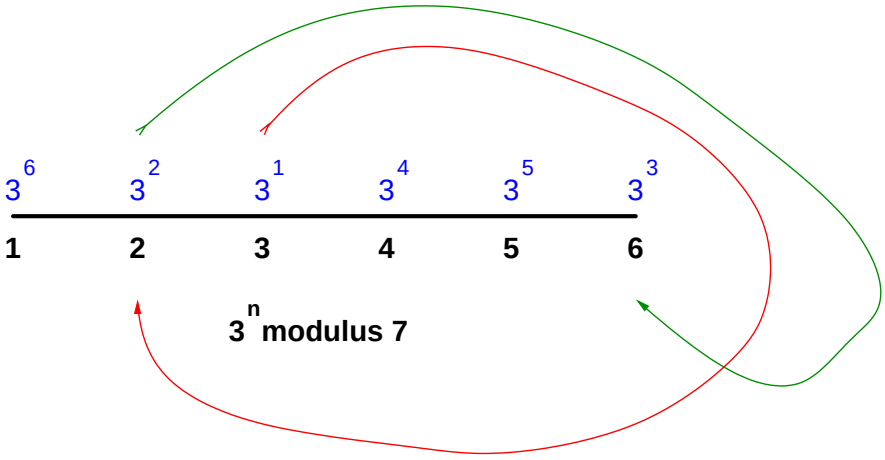
Alice	Bob
agree on $g=6$ and $p=7$	agree on $g=6$ and $p=7$
generate random $x=5$	generate random $y=4$
compute $(6+5) \bmod 7 = 4$	compute $(6+4) \bmod 7 = 3$
send $4 \rightarrow$	$\leftarrow$ send 3
receive $3 \leftarrow$	$\rightarrow$ receive 4
use 3, compute $(4 + 3) \bmod 7 = 0$	use 4, compute $(3 + 4) \bmod 7 = 0$
use as shared key 0	use as shared key 0

Real Key Exchange Example

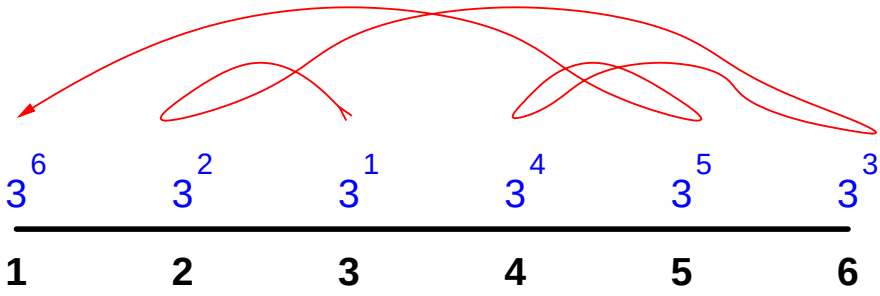
This uses very small values:

Alice	Bob
agree on g=*3* and p=*7*	agree on g=*3* and p=*7*
generate random x=*3*	generate random y=*4*
compute $3^3 \ mod \ 7 = 6$	compute $3^4 \ mod \ 7 = 4$
send *6* $\rightarrow$	$\leftarrow$ send *4*
receive *4* $\leftarrow$	$\rightarrow$ receive *6*
use *4* to compute $6^4 \ mod \ 7 = 1$	use *6* to compute $4^6 \ mod \ p = 1$
use as shared key *1*	use as shared key *1*

Unpredictability of Modulus Exponentiation

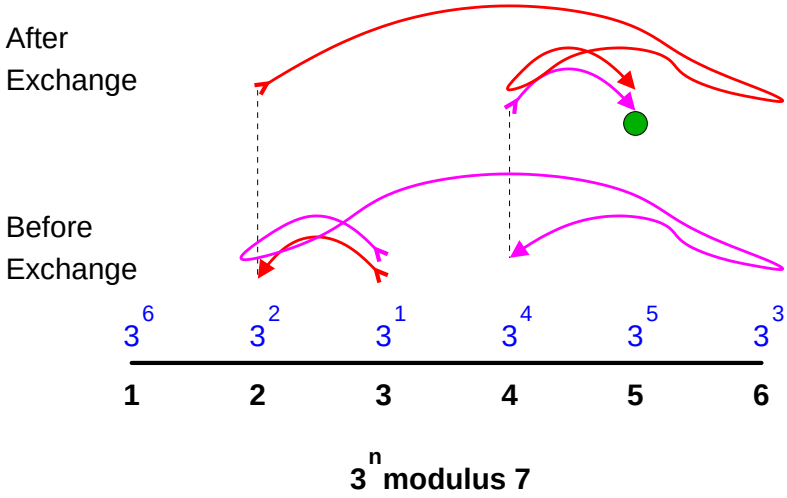


Unpredictability of Modulus Exponentiation



While it is computationally easy to compute higher exponents from a given modulus, it is computationally impossible to compute lesser exponent moduli (a trapdoor function).

Both Arrive at the Same Number



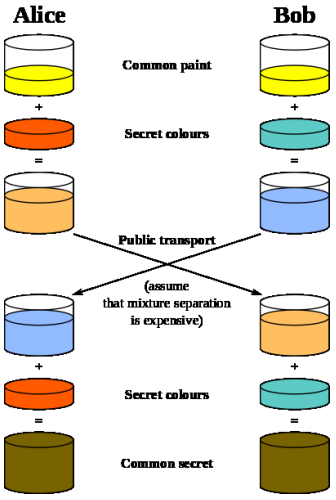
Discrete Logarithm in a Finite Field

If an eavesdropper sees $g^x \; mod \; p$, how do they find x, e.g.,

$$\begin{aligned} 3^x &= 729 \\ x \times log\, 3 &= log\, 729 \\ x &= \frac{log\, 729}{log\, 3} \\ x &= 6 \\ (x \times log\, 3) \; mod \; p &\neq (log\, 729) \; mod \; p \end{aligned}$$

Modulus a prime p creates a finite (Galois) field where logarithms don't work; see https://en.wikipedia.org/wiki/Finite_field.

Diffie–Hellman Illustrated with Paint



https://commons.wikimedia.org/wiki/File:Diffie-Hellman_Key_Exchange.svg

Elliptic Curve Diffie–Hellman (ECDHE)

Similar to normal Diffie-Hellman because:

- A generator g and modulus-prime p are used, and are public
- Computation commutativity is used by each participant to privately compute the secret key

but it is different because:

- Values exist as points on an elliptic curve, rather than being linear (on a number line)
- g is an ordered-pair of integers (a 2D point), not a single integer

$$y^2 = x^3 + ax + b$$

Public constants a and b define the elliptic curve. Elliptic curves are Abelian groups, allowing commutativity; see https://en.wikipedia.org/wiki/Abelian_group.

ECDHE Exchange

- The participants each generate a random number, like DHE
- Starting from the generator point, they use point doubling and addition to simulate moving the point their random number of times
- They transmit their new points to each other
- Using the received points they double/add them their random number of times
- Both participants end at the same point on the elliptical curve
- They use the x component of the result to derive a shared secret

Elliptic Curve Doubling and Addition

Suppose the random number is *71* (binary 1000111) . We need to compute the effect of starting at point g and advancing 70 more times along the elliptic curve. Doubling and addition can be used to shorten the computation to reach the final point:

$$
\begin{aligned}
2 \times g &= 2g \\
2 \times 2g &= 4g \\
2 \times 4g &= 8g \\
2 \times 8g &= 16g \\
16g + g &= 17g \\
2 \times 17g &= 34g \\
34g + g &= 35g \\
2 \times 35g &= 70g \\
70g + g &= 71g
\end{aligned}
$$

Efficient Exponentiation

Elliptic curve doubling and addition is similar to *exponentiation by squaring* (https://en.wikipedia.org/wiki/Exponentiation_by_squaring), used by DHE and RSA to efficiently raise integers to large integer powers. Modulus reduction can happen at each stage.

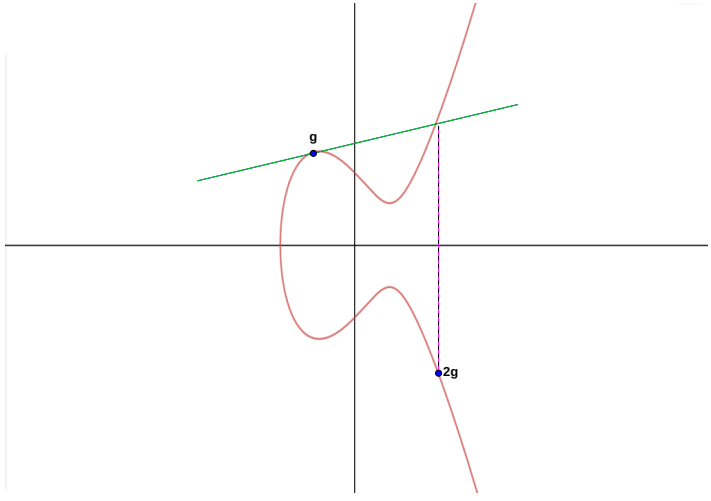
ECDHE Point Computations

- Start with the generator point g
- Double it to get $2g$
- Doubling involves computing a line tangent to the point being doubled and finding the line intersection point with the elliptical curve
- This is effectively adding a point to itself, assuming the two points are on top of each other
- Reflect the point across the x axis
- Addition involves drawing a line between two points and computing the intersection point, then reflecting

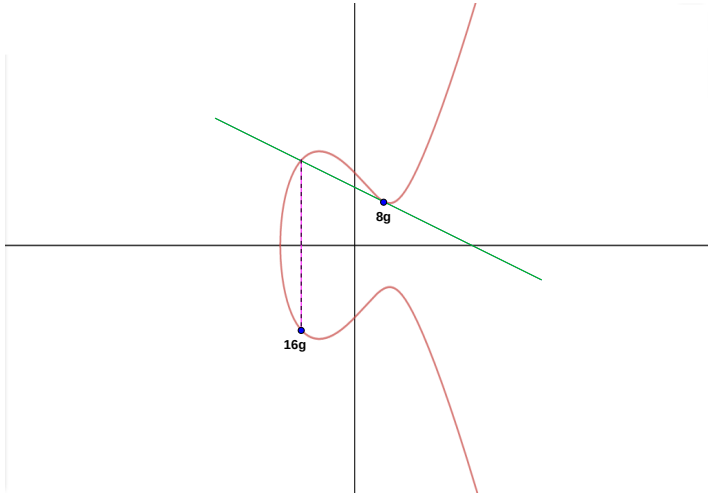
=

http://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/

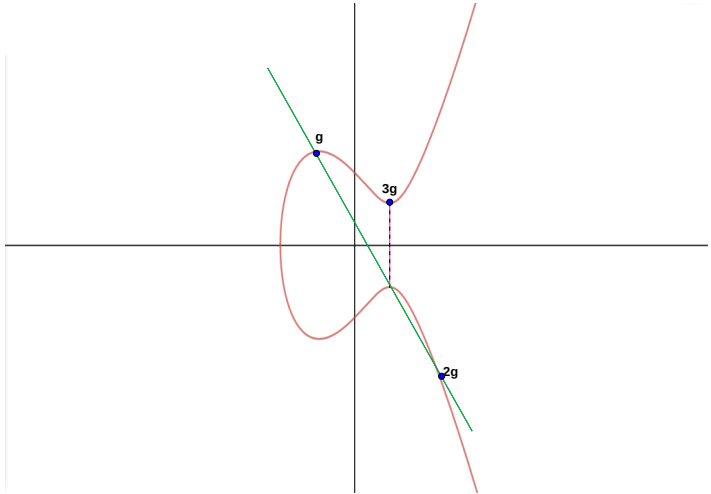
Elliptic Curve Doubling g



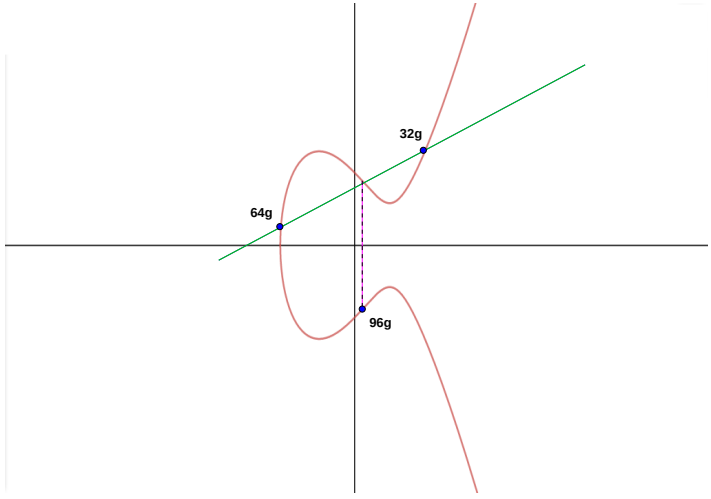
Elliptic Curve Doubling $8g$



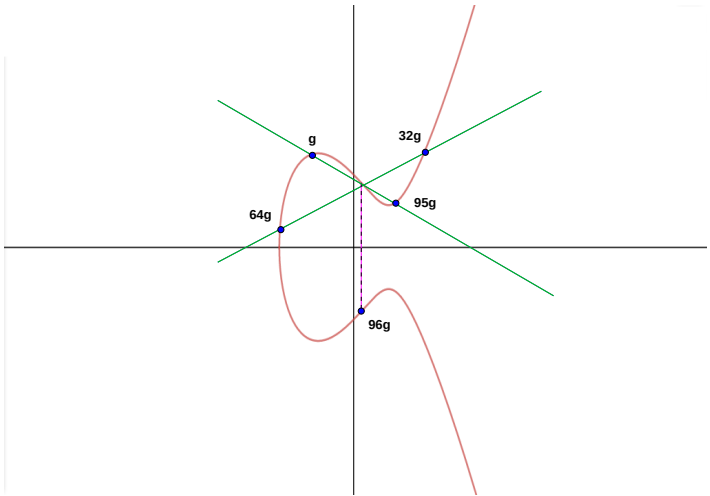
Elliptic Curve Adding g and $2g$



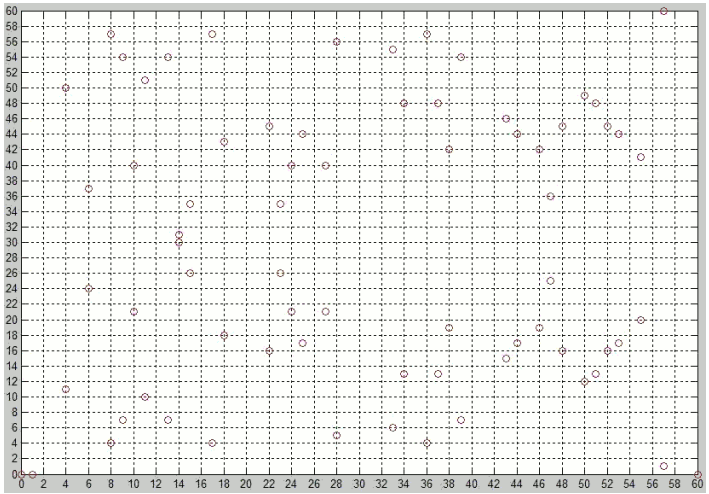
Elliptic Curve Adding $64g$ and $32g$



Adding One vs. Adding Larger Numbers



Elliptic Curve Distribution for Prime of 61



Modulus forces the values into a finite field.

https://hu.wikipedia.org/wiki/Elliptikus_g\%C3\%B6rbe\#/media/File:Elliptic_curve_y\%5E2\%3Dx\%5E3-x_on_finite_field_Z_61.PNG

ECDHE Conclusion

Elliptical curve Diffie-Hellman requires less computation than DHE, e.g., 224-bit ECDHE is as strong as 2048-bit DHE.
For an elliptical curve cryptography overview watch the first half of the video by Martijn Grooten, *https://youtu.be/yBr3Q6xiTw4*, and for details see the video by Nigel Smart, https://youtu.be/t3JzdKE-Fhs.

5 Public/Private-Key Communication: RSA (Rivest, Shamir, Adleman)

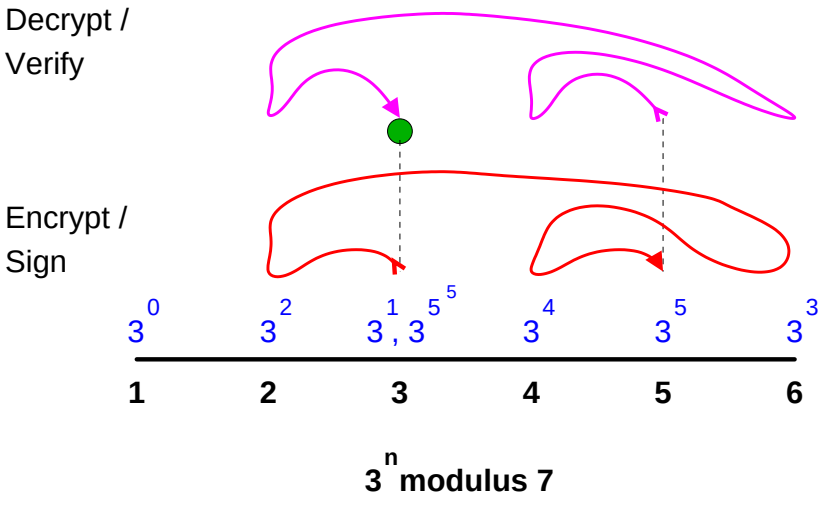
Diffie–Hellman:

- Negotiates a shared secret
- Uses a public exponent base g
- Uses a public modulus prime p
- Each client generates a secret exponent; is peer-to-peer key negotiation

RSA:

- Communicates a message
- The message is a base value, raised to an exponent
- The public modulus n is the product of two private primes, p and q
- One exponent e is public, another d is private
- Raising a message to the power e and then d (or the reverse order) returns the original message
- One party controls the private key
- Performs encryption/decryption and signing/verification

Using Two Exponents to Return to the Origin



Notice decrypt/verify exponentiates the output of encrypt/sign, not the base value, which is not known at that stage.

Exponentiation p Returns the Base

Fermat's Little Theorem, year 1640, states that for all m in the range $1 < m < p$ and p prime:

$$\begin{aligned} m^{p-1} \bmod p &= 1 \\ m \times (m^{p-1} \bmod p) &= m \\ (m \times m^{p-1}) \bmod p &= m \\ m^p \bmod p &= m \end{aligned}$$

This allows large exponentiation in a modulus field to return the base value:

- https://mathlesstraveled.com/2017/10/14/four-formats-for-fermat/
- https://mathlesstraveled.com/2017/11/13/fermats-little-theorem-proof-by-modular-arithmetic/
- https://en.wikipedia.org/wiki/Proofs_of_Fermat%27s_little_theorem

Exponentiation p - 1 Can Be Applied Multiple Times

Exponentiation in a modulus field is cyclic, so multiple exponent values return the same result. For integer k

$$\begin{aligned} m^{p-1} \bmod p &= 1 \\ m^{k\times(p-1)} \bmod p &= 1 \\ m^{k\times(p-1)+1} \bmod p &= m \end{aligned}$$

Using Two Primes to Return the Base

Euler's Theorem, year 1763, states that for primes p and q, and $lcm()$ as the least common multiple:

$$\begin{aligned} m^{p-1} \bmod p &= 1 \\ m^{(p-1)(q-1)} \bmod (p \times q) &= 1 \\ m^{lcm((p-1)(q-1))} \bmod (p \times q) &= 1 \\ m^{lcm((p-1)(q-1))+1} \bmod (p \times q) &= m \end{aligned}$$

- https://en.wikipedia.org/wiki/Euler%27s_theorem
- https://en.wikipedia.org/wiki/Euler%27s_totient_function

Least Common Multiple Can Be Applied Multiple Times

For integer k:

$$
\begin{aligned}
m^{lcm((p-1)(q-1))+1} \ mod \ (p \times q) &= m \\
m^{k \times lcm((p-1)(q-1))+1} \ mod \ (p \times q) &= m
\end{aligned}
$$

Creating a Public/Private Exponent Pair

Instead of using p and q directly to compute an exponent pair that will return m, choose a small exponent e and compute an exponent d that will return m:

$$\begin{aligned} e \times d &= k \times lcm((p-1)(q-1)) + 1 \\ (e \times d)\ mod\ lcm((p-1)(q-1)) &= 1 \\ d &= \frac{1}{e}\ mod\ lcm((p-1)(q-1)) * \end{aligned}$$

* Computing a fraction in a modulus field is accomplished using the Chinese remainder theorem, https://en.wikipedia.org/wiki/Chinese_remainder_theorem.

Exponentiation Using d and e

$$
\begin{aligned}
m^{k \times lcm((p-1)(q-1))+1} \; mod \; (p \times q) &= m \\
d \times e &= k \times lcm((p-1)(q-1)) + \\
m^{e \times d} \; mod \; (p \times q) &= m \\
m^{e^d} \; mod \; (p \times q) &= m \\
m^{d^e} \; mod \; (p \times q) &= m
\end{aligned}
$$

Intermediate Modulus Operations

$$
\begin{aligned}
m^{e^d} \bmod (p \times q) &= m \\
m^{d^e} \bmod (p \times q) &= m \\
(m^e \bmod (p \times q))^d \bmod (p \times q) &= m \\
(m^d \bmod (p \times q))^e \bmod (p \times q) &= m
\end{aligned}
$$

Summary of Exponentiation With Modulus

- Base values raised to specific exponents with modulus return the original values, $m^p \bmod p = m$
- Multiple exponents can do this, $m^{k\times(p-1)+1} \bmod p = m$
- When using a modulus that is the product of two primes, finding an exponent that returns the base value is hard to compute using just the modulus, $m^{lcm((p-1)(q-1))+1} \bmod (p \times q) = m$
- Multiple exponents can do this, $m^{k\times lcm((p-1)(q-1))+1} \bmod (p \times q) = m$
- Other exponent pairs, which are more distantly related to the two primes, can also do this, $e \times d = k \times lcm((p-1)(q-1)) + 1$
- Exponent pairs can be applied in any order, $m^{e^d} \bmod (p \times q) = m^{d^e} \bmod (p \times q)$
- Modulus can be applied to intermediate results, $(m^e \bmod (p \times q))^d \bmod (p \times q) = m$

Public/Private Designations

Public

- $n = p \; x \; q$, used as a modulus
- exponent e

Private

- exponent d
- primes p and q
- $(p - 1)(q - 1)$, used to compute d

The RSA key length indicates the bits in n, e.g., 2048 bits.
Modulus is used to compute the exponent d and applied to the exponentiation result.

$$g^{y^x} \; mod \; p \quad = \quad g^{x^y} \; mod \; p$$

RSA Application

Dual-exponent ordering allows for encryption and signing (authentication):

$$\begin{aligned} \text{encrypt}\ \ message^{e} \bmod n &= encrypted \\ \text{decrypt}\ \ encrypted^{d} \bmod n &= message \end{aligned}$$

$$\begin{aligned} \text{sign}\ \ hash(message)^{d} \bmod n &= signature \\ \text{verify}\ \ signature^{e} \bmod n &= hash(message) \end{aligned}$$

Encryption uses the public exponent e to encrypt a message to produce encrypted text that is later decrypted with private exponent d. Signing uses the private exponent d to encrypt the hash of a message to produce a signature which is later validated by applying the public exponent e. Signing the hash of Diffie–Hellman parameters authenticates the parameters as being created by the private key owner.

6. Conclusion: What Cryptography Can Accomplish

Confidentiality Only the other party can read the original messages (ECDHE, AES)

Authenticity Verify who is on the other end of the communication channel (RSA via X.509 certificates)

Integrity No other party can change or add messages (GCM, SHA)

Further Cryptography Study

For additional study about cryptography, watch the nine video lessons at https://www.youtube.com/playlist?list=PLqhpVxkBo1dPiKHym2CxOKEnqCO35OJH2. The videos were created by Bill Buchanan, Professor in the School of Computing at Edinburgh Napier University. His website, http://asecuritysite.com/, contains a wealth of information about digital security.

Further Cryptography Reading

- Niels Ferguson, Bruce Schneier, Tadayoshi Kohno, *Cryptography Engineering: Design Principles and Practical Applications,* 2010: practical overview of cryptographic protocols
- Christof Paar, Jan Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners,* 2010: scholarly overview of all modern cryptographic protocols
- Bruce Schneier, *Applied Cryptography,* 1996: historical but broad overview of the cryptographic landscape

Conclusion

https://momjian.us/presentations

https://www.flickr.com/photos/vmax137/