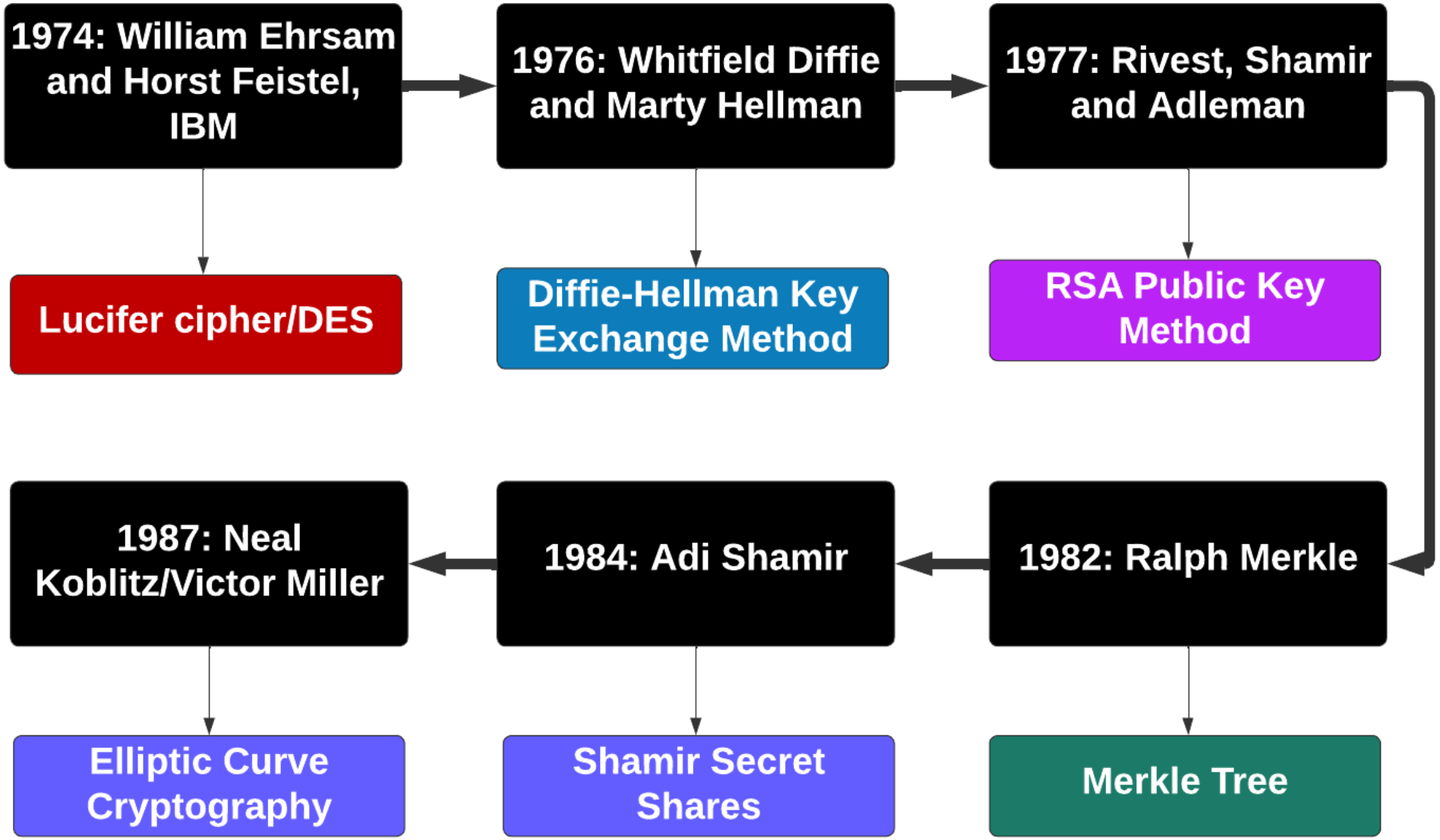
Cryptography

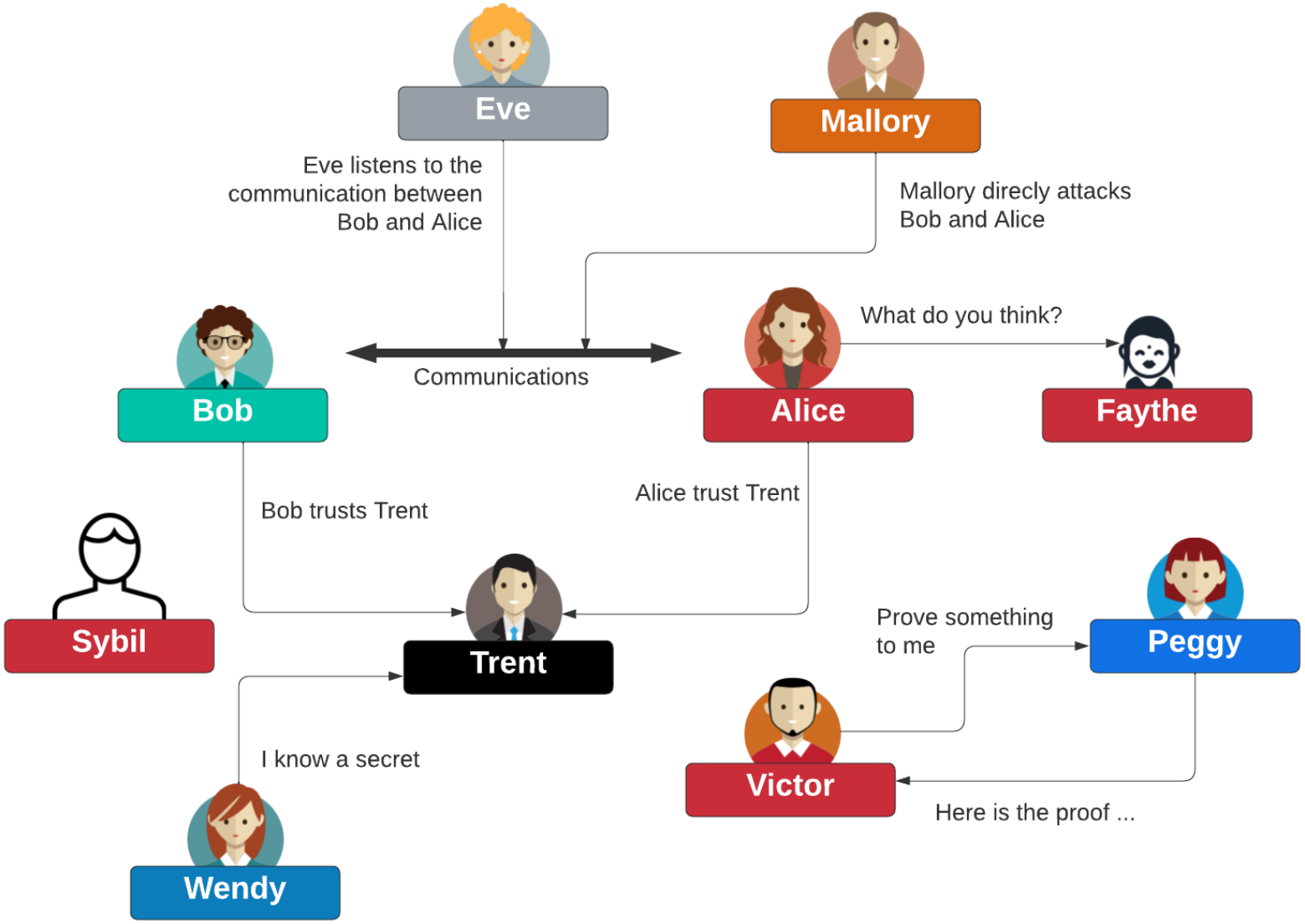
Prof Bill Buchanan OBE FRSE

https://asecuritysite.com

A Bit of History



Some Actors


Eve
Mallory
Eve listens to the communication between Bob and Alice
Mallory direcly attacks Bob and Alice
Bob
Communications
Alice
What do you think?
Faythe
Bob trusts Trent
Alice trust Trent
Sybil
Trent
Prove something to me
Peggy
Victor
I know a secret
Here is the proof ...
Wendy

Cryptography: Fundamentals

Prof Bill Buchanan OBE FRSE

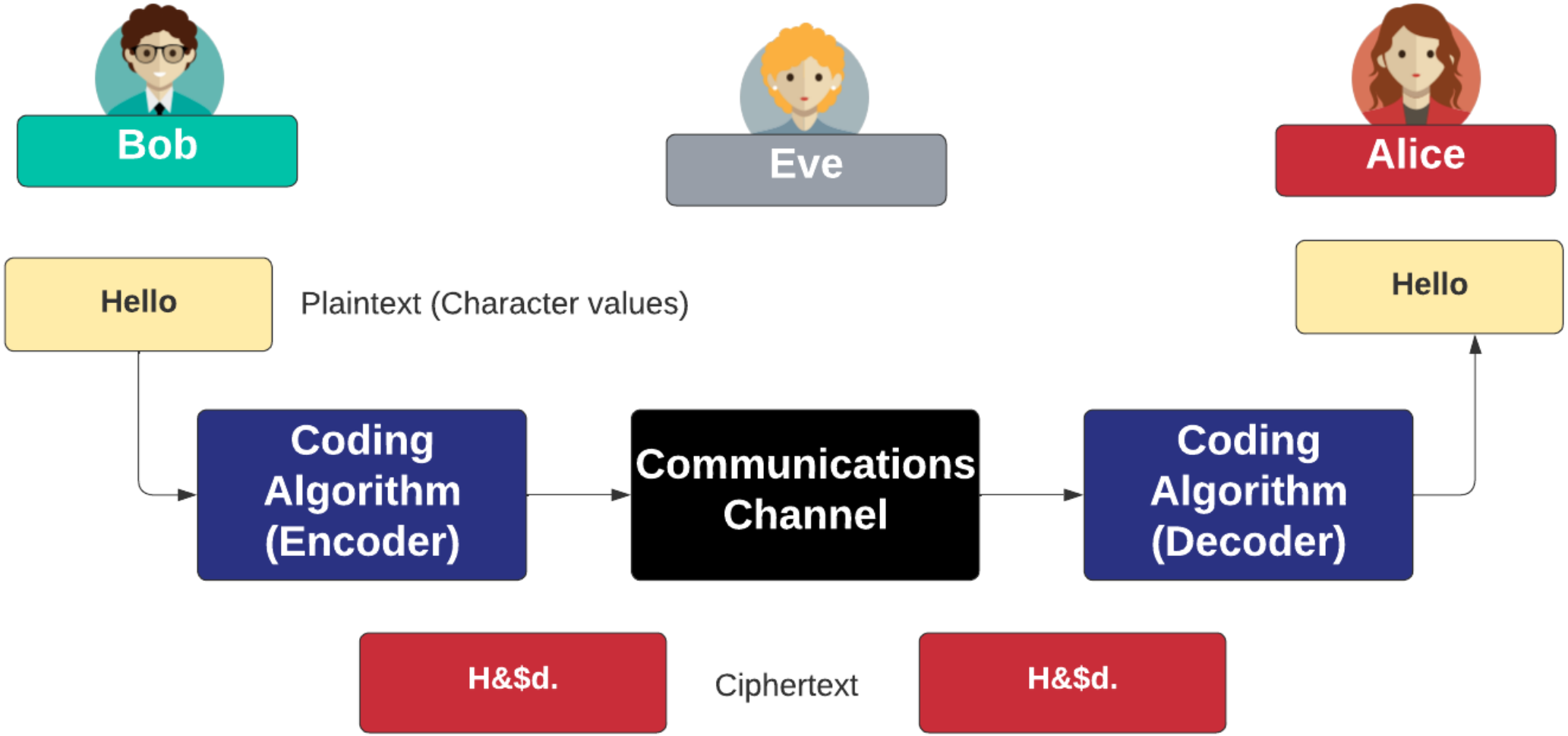
https://asecuritysite.com

Fundamentals
Symmetric Key
Hashing
MAC
KDF
Public Key
Key Exchange
Signatures
Digital Certificates

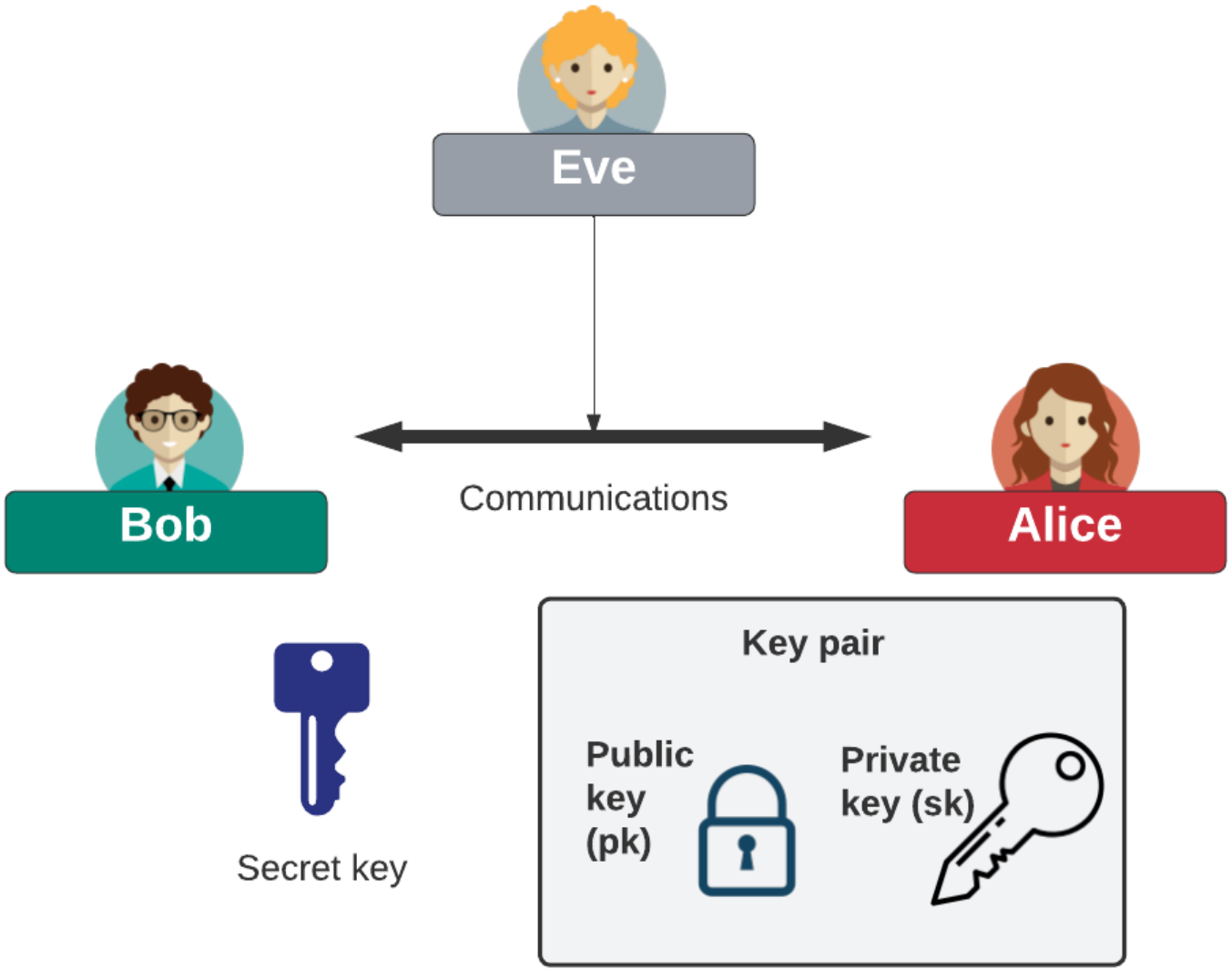
BLOCKPASS IDENTITY LAB

World-leading Collaboration between
Blockpass IDN and Edinburgh Napier University

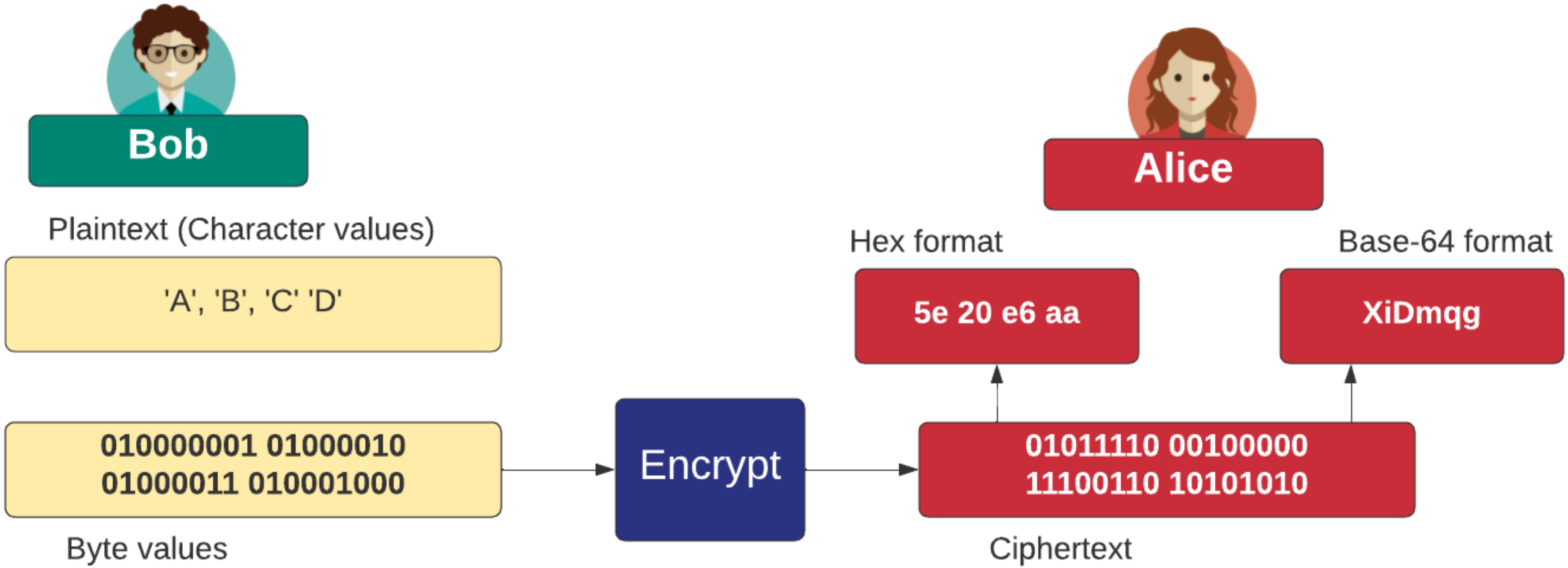
Encoding and Decoding



Keys


Eve
Bob
Communications
Alice
Secret key
Key pair
Public key (pk)
Private key (sk)

Encoding


Bob
Alice
Plaintext (Character values)
'A', 'B', 'C' 'D'
Hex format
Base-64 format
5e 20 e6 aa
XiDmqg
010000001 01000010
01000011 010001000
Encrypt
01011110 00100000
11100110 10101010
Byte values
Ciphertext

Hexadecimal

Bob

For hex convertion, take the bit stream, and split into groups of four bits

Bit stream

0101 1110 0010 0000 1110 0110 1010 1010

Hex

5 e 2 0 e 6 a a

5e | 20 | e6 | aa

Bytes

Value	Bit value
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Base 64

For Base-64, we take the bit stream and convert into groups of six bits

Bit stream

01100110 01110010 01100101 01100100

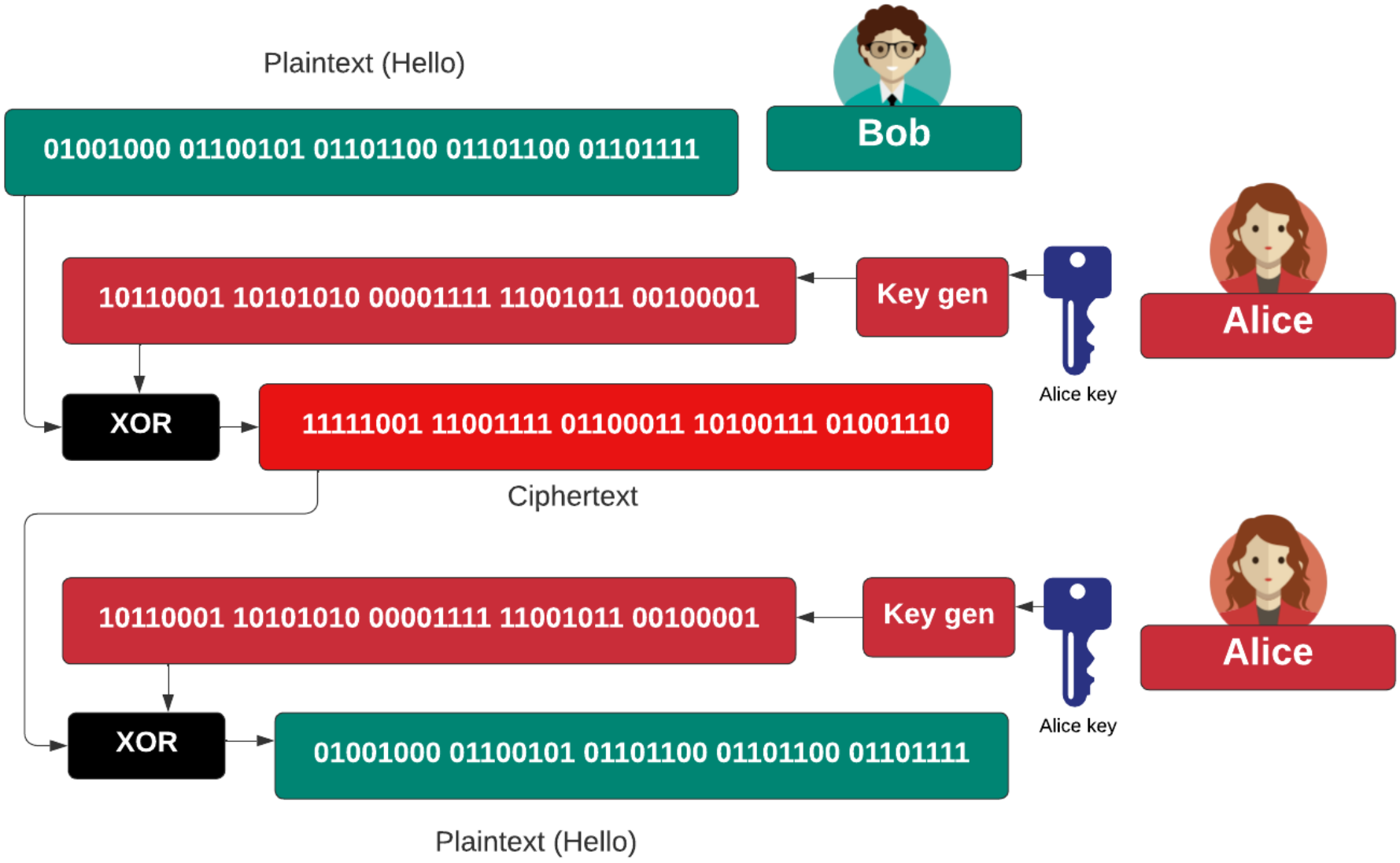
011001 100111 001001 100101 011001 00

Base64

Z n J l Z A = =

Val	Enc	Val	Enc	Val	Enc	Val	Enc
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

A Basic Operation



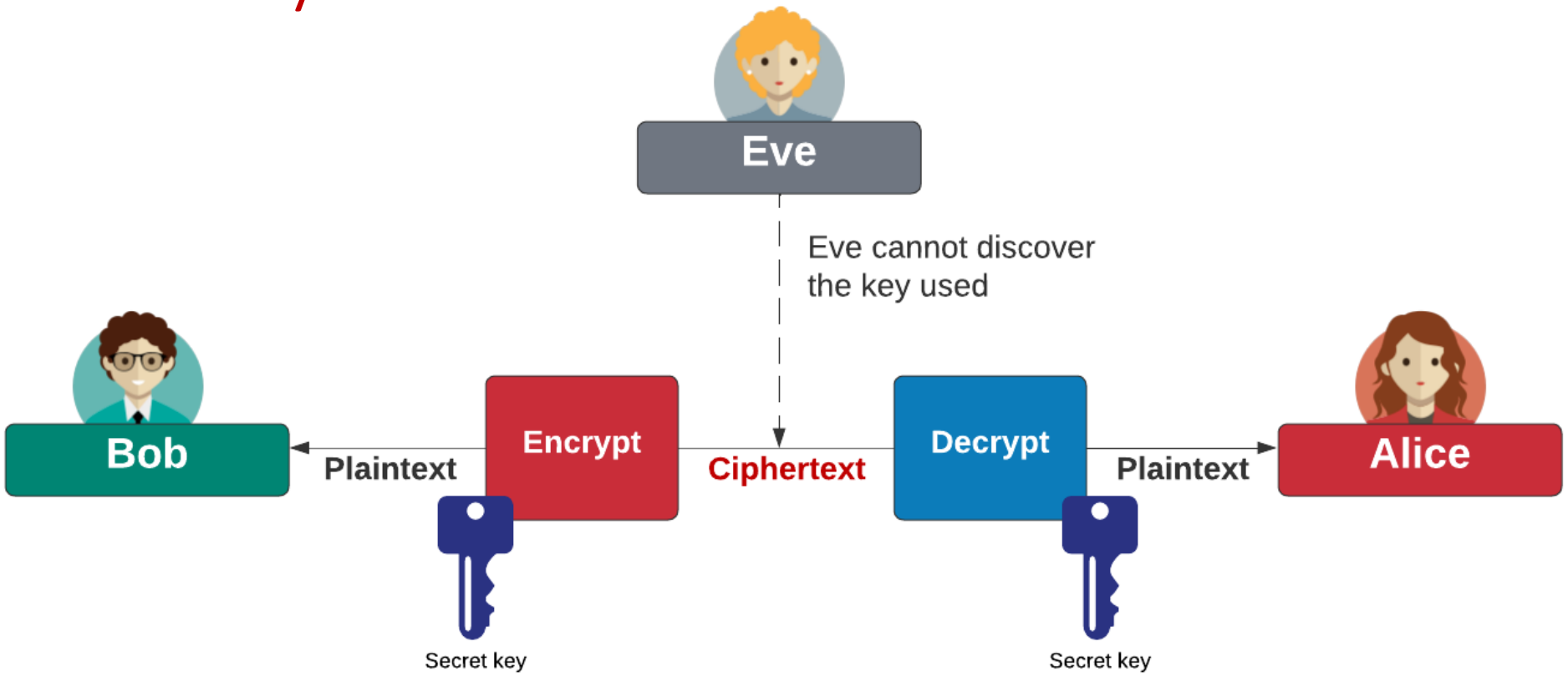
Cryptography: Symmetric Key

Prof Bill Buchanan OBE FRSE

https://asecuritysite.com/symmetric

Fundamentals
Symmetric Key
Hashing
MAC
KDF
Public Key
Key Exchange
Signatures
Digital Certificates

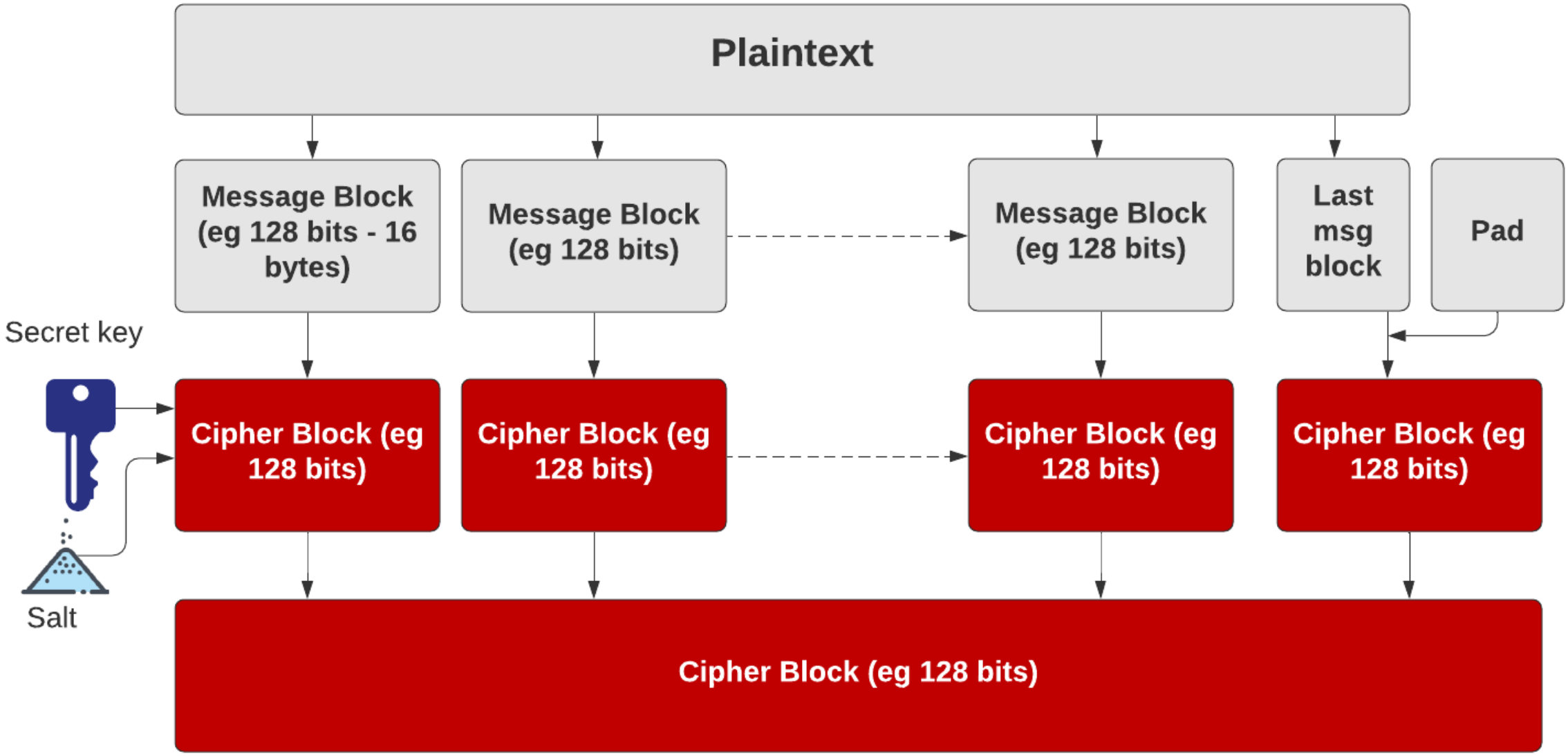
Symmetric Key



Block cipher: **AES (128-bit block)**, DES (64-bit block), 3DES (64-bit block), Blowfish, SM4, Camellia, GOST, IDEA, RC2, Serpent, Skipjack, Twofish, and XTEA
Stream cipher: **ChaCha20**, **AES GCM**, ARIA, Sala20, and RC4.

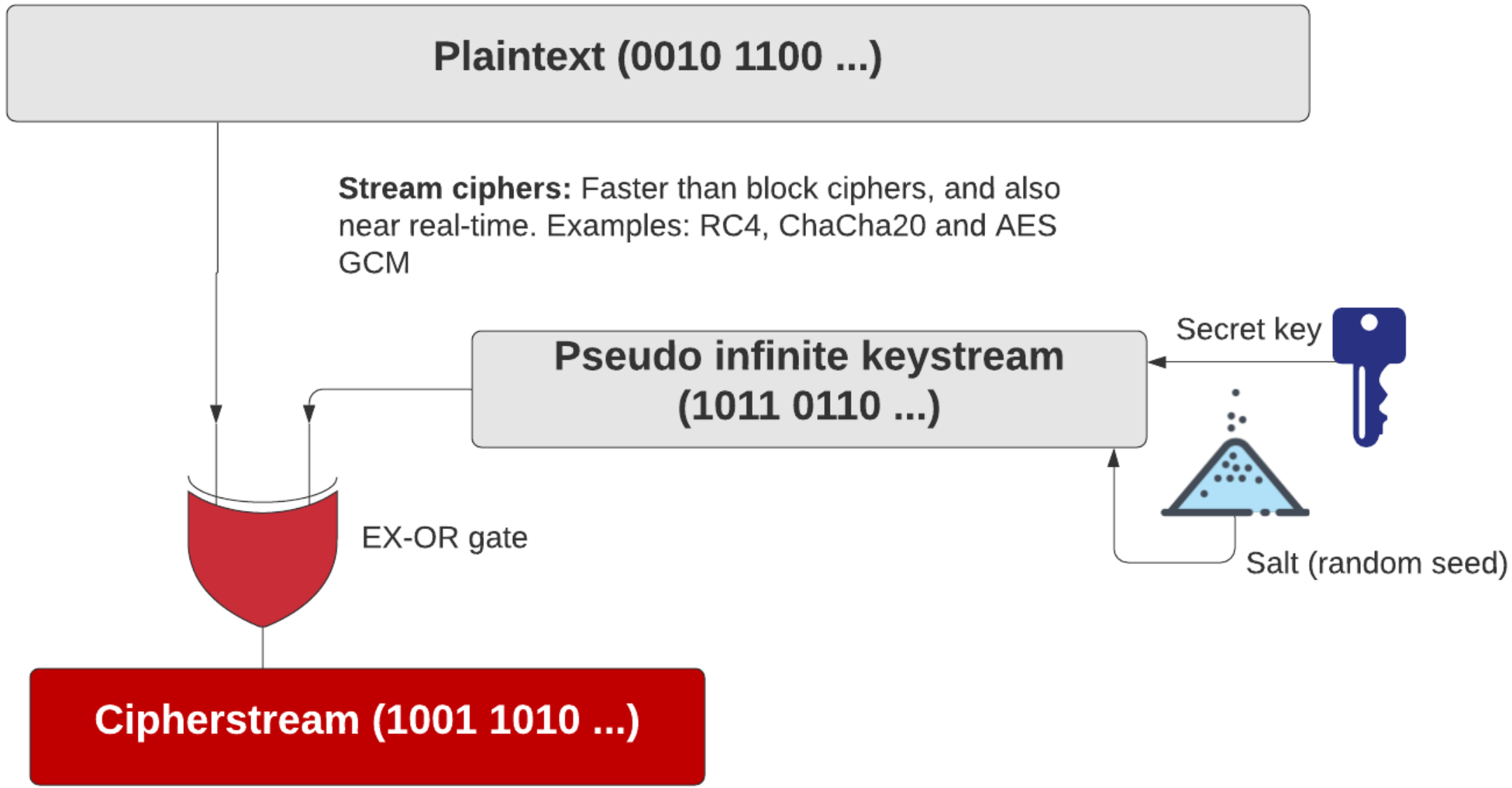
AES also known as Rijndael (after Joan Daemen and Vincent Rijmen).
ChaCha20 created by Daniel J Bernstein.
Blowfish and Twofish created by Bruce Schneier.

Symmetric Key – Block cipher



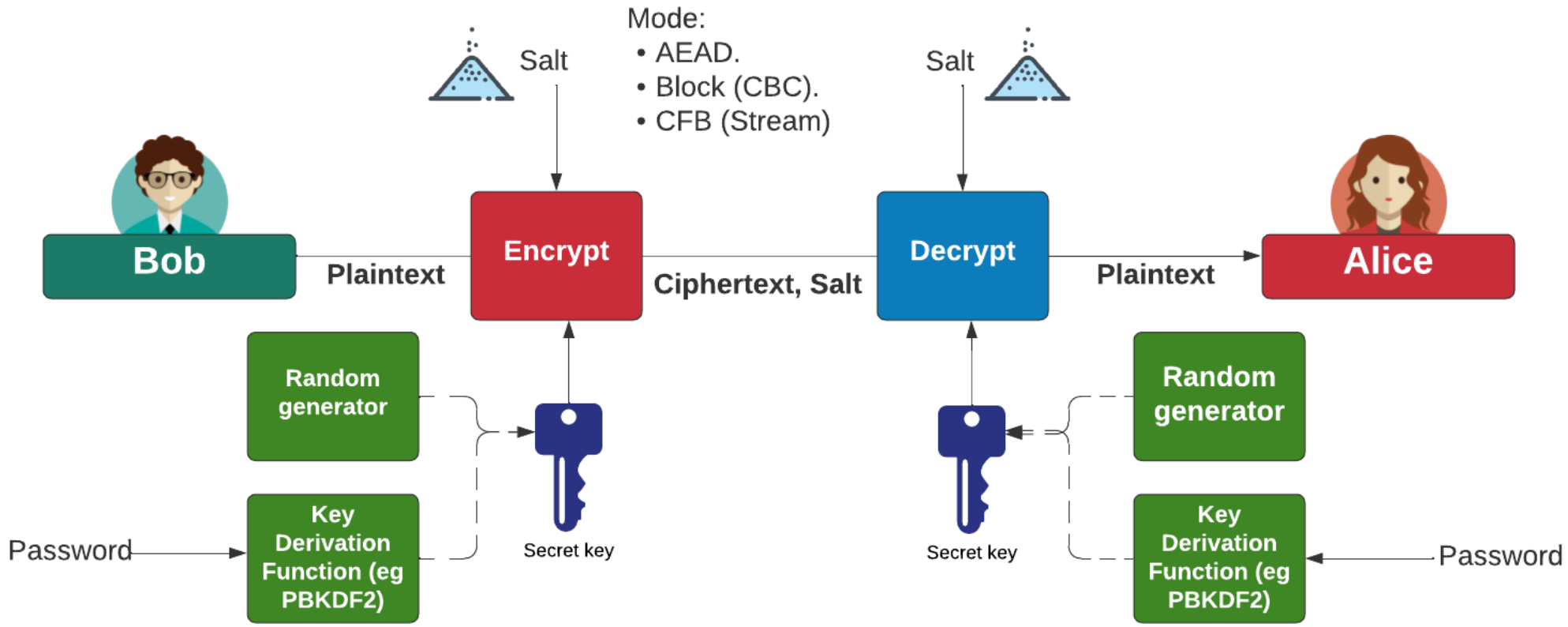
https://asecuritysite.com/golang/go_pad

Symmetric Key – Stream cipher



https://asecuritysite.com/golang/go_chacha

Symmetric Key - Modes

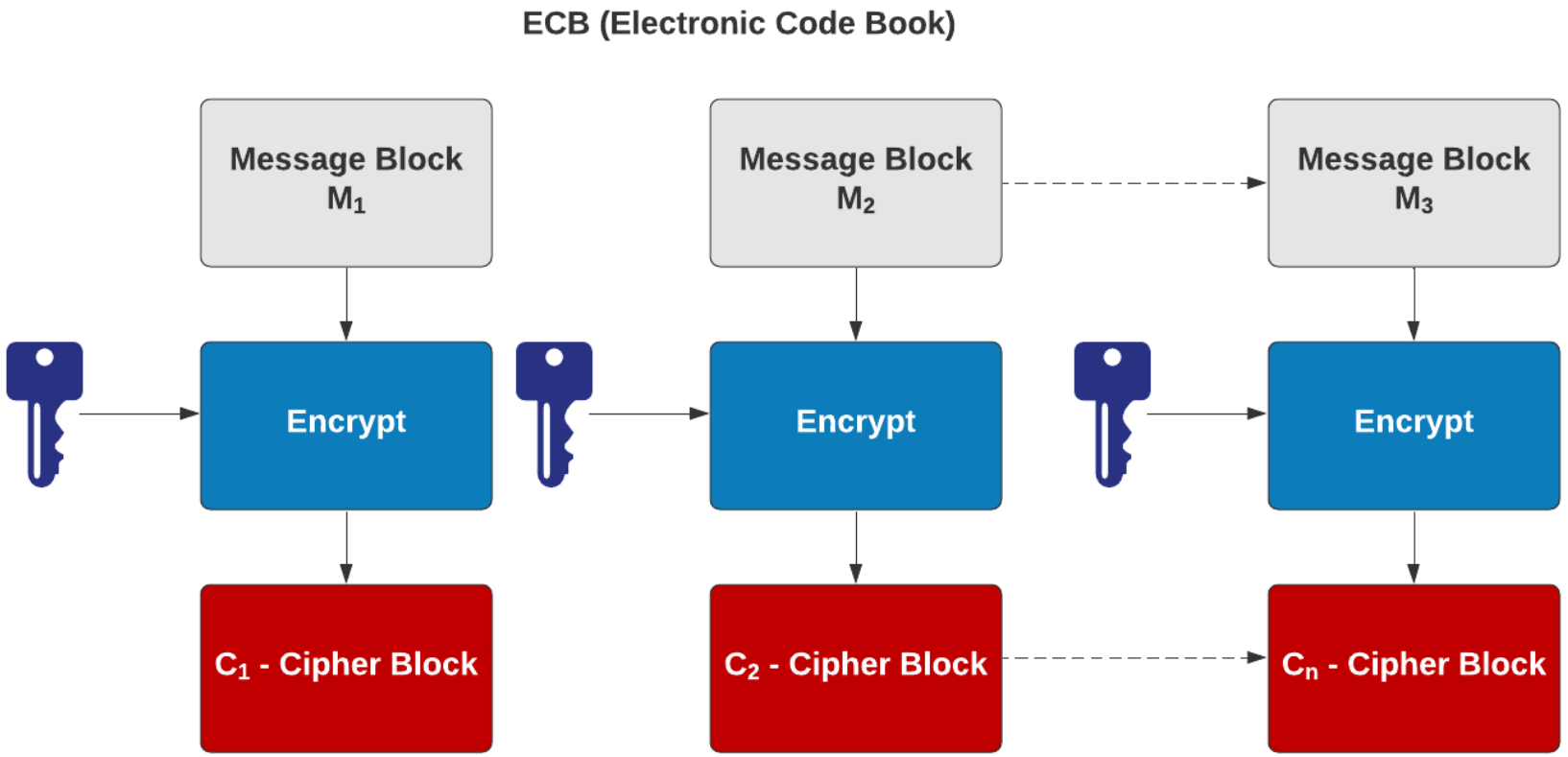


AES Block cipher modes: ECB (No salt), CBC (Cipher Block Chaining).

AES Stream modes: CFB (Cipher Feedback), OFB (Output Feedback), CTR (Counter), **GCM** (Galois Counter Mode).

https://asecuritysite.com/golang/go_symmetric

Symmetric Key – Electronic Code Code (ECB)

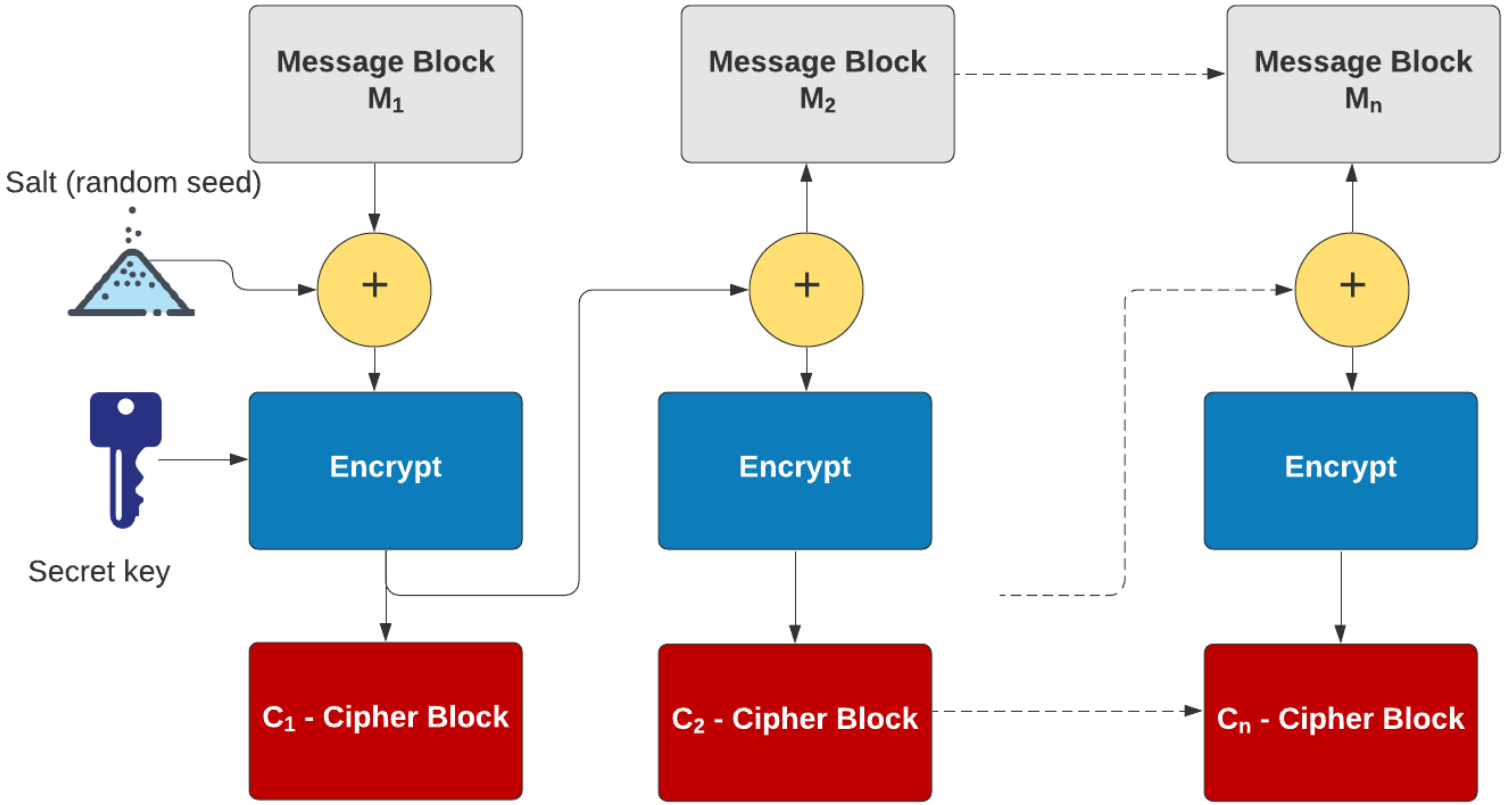


AES Block cipher modes: ECB (No salt), CBC (Cipher Block Chaining).

AES Stream modes: CFB (Cipher Feedback), OFB (Output Feedback), CTR (Counter), GCM (Galois Counter Mode).

https://asecuritysite.com/hazmat/hashnew4

Symmetric Key – Cipher Block Chaining (CBC)

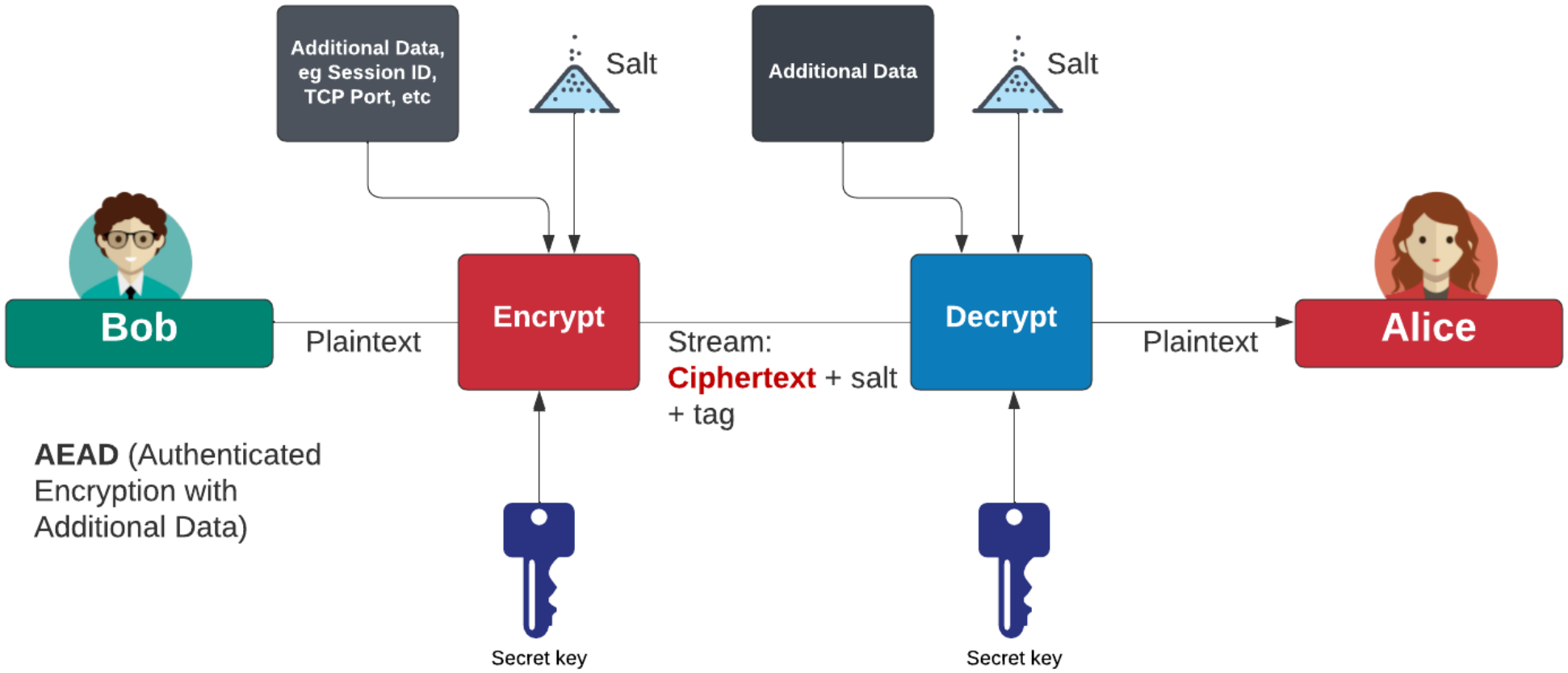


AES Block cipher modes: ECB (No salt), CBC (Cipher Block Chaining).

AES Stream modes: CFB (Cipher Feedback), OFB (Output Feedback), CTR (Counter), GCM (Galois Counter Mode).

https://asecuritysite.com/golang/go_symmetric

Authenticated Encryption with Additional Data (AEAD)



AES Block cipher modes: ECB (No salt), CBC (Cipher Block Chaining).

AES Stream modes: CFB (Cipher Feedback), OFB (Output Feedback), CTR (Counter), GCM (Galois Counter Mode).

https://asecuritysite.com/golang/go_symmetric

Cryptography: Hashing

Prof Bill Buchanan OBE FRSE

https://asecuritysite.com/hashing

Fundamentals
Symmetric Key
Hashing
MAC
KDF
Public Key
Key Exchange
Signatures
Digital Certificates

BLOCKPASS IDENTITY LAB
World-leading Collaboration between
Blockpass IDN and Edinburgh Napier University

Hashing – MD5 (128-bit)

Hex

hello	5D41402ABC4B2A76B9719D911017C592
Hello	8B1A9953C4611296A827ABF8C47804D7
How are you?	04E35EB3E4FCB8B395191053C359CA0E
Napier	8F83571F9324AE4E23D773753055C7B6

Base64

hello	XUFAKrxLKna5cZ2REBfFkg==
Hello	ixqZU8RhEpaoJ6v4xHgE1w==
How are you?	BONes+T8uLOVGRBTw1nKDg==
Napier	j4NXH5Mkrk4j13N1MFXHtg==

https://asecuritysite.com/hash/md5

Hashing – SHA-1 (160-bit)

SHA-1 Hash
(160-bit hash)

Hex

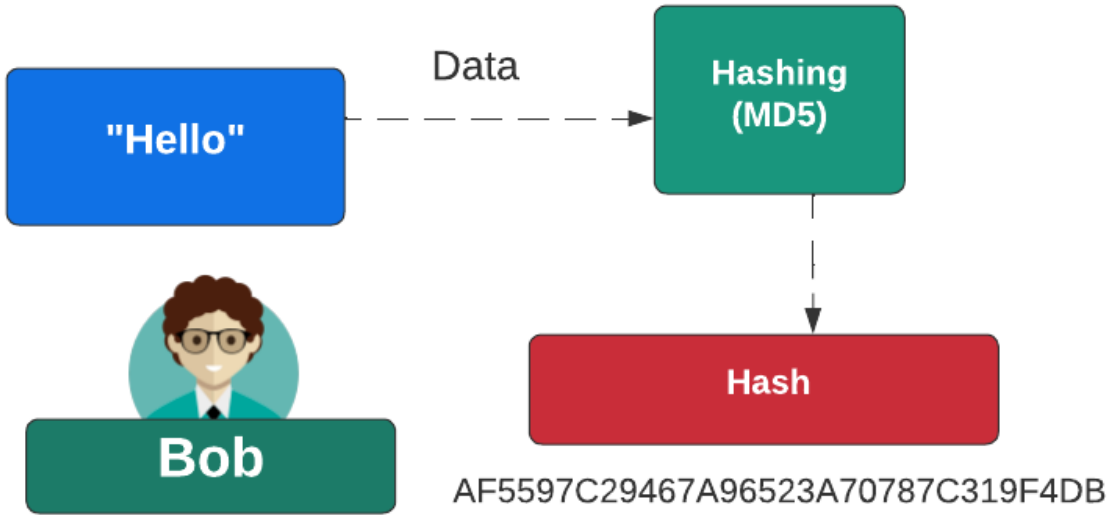
hello	AAF4C61DDCC5E8A2DABEDE0F3B482CD9AEA9434D
Hello	F7FF9E8B7BB2E09B70935A5D785E0CC5D9D0ABF0
How are you?	3031897E282167593FBB4DBE81DC48EBBE9A002D
Napier	BF81B135A5687766F4F464764EAC38CA8A4EBABA

Base64

hello	qvTGHdzF6KLavt4PO0gs2a6pQ00=
Hello	9/+ei3uy4Jtwk1pdeF4MxdnQq/A=
How are you?	MDGJfighZ1k/u02+gdxl676aAC0=
Napier	v4GxNaVod2b09GR2Tqw4yopOuro=

https://asecuritysite.com/hash/md5

Password Hashing



Hashed password

Bob: AF5597C29467A96523A70787C319F4DB

Hashed password with salt (**zj8n**hello)

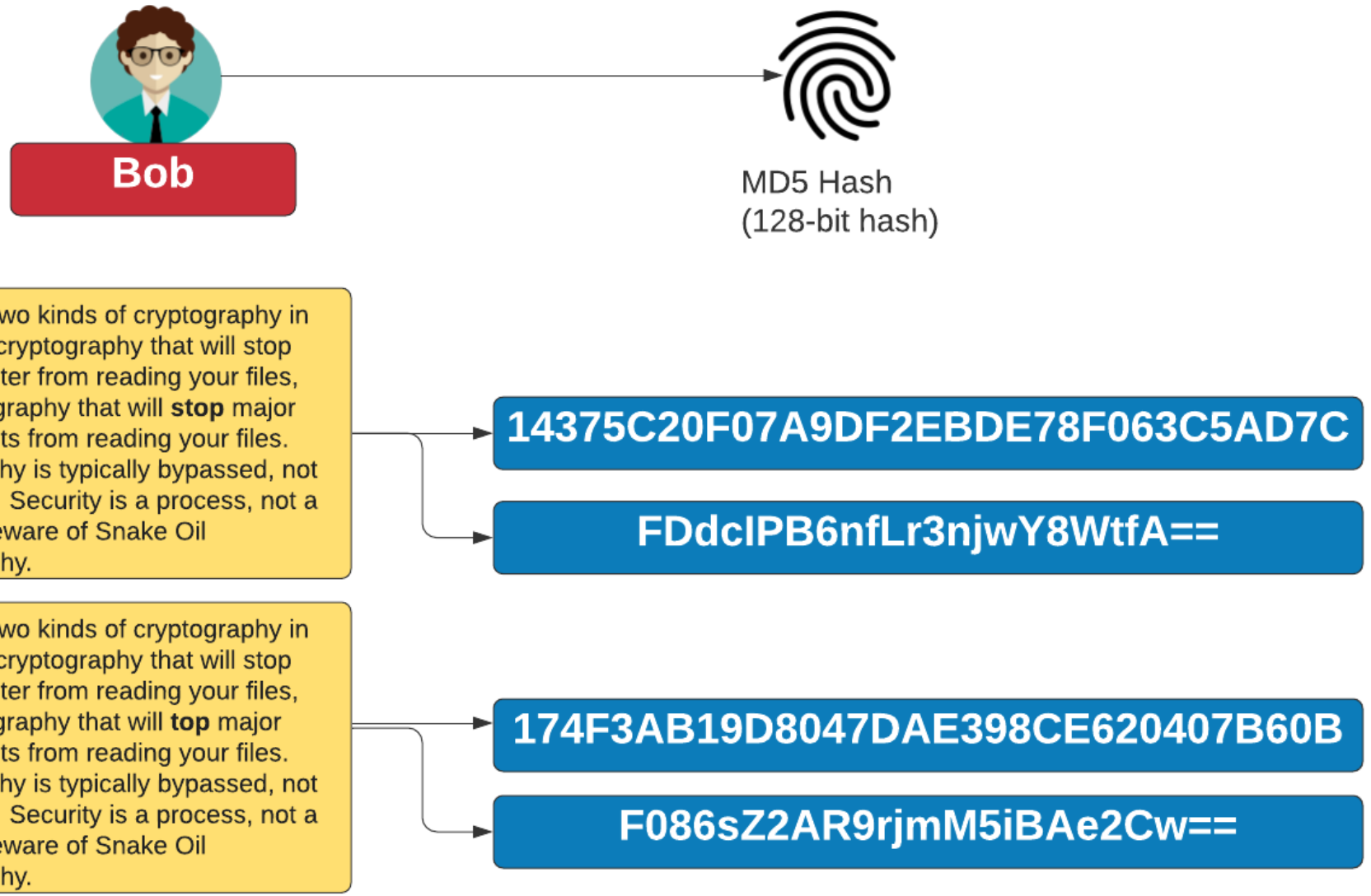
Bob:**zj8n**:51A7C663A3BDCD06D6CE21E2BCB2AD5A

 # openssl passwd -1 -s abc Hello
Word: Hello
Method: -1
Salt: **abc**
$**1**$**abc**$0lWzLRntj3ldjLzhKriFx1

Cryptographic Hashes: MD5, SHA-1, SHA-256, SHA-3, RIPEMD, Blake2b
Windows hashes: NTLM
Non-cryptographic hashes: Spooky, Farm, FVN

https://asecuritysite.com/golang/gohash https://asecuritysite.com/openssl/passwds

Hashing



https://asecuritysite.com/hash/hashnew

Hashing and OpenSSL

There are two kinds of cryptography in this world: cryptography that will stop your kid sister from reading your files, and cryptography that will **stop** major governments from reading your files. Cryptography is typically bypassed, not penetrated. Security is a process, not a product. Beware of Snake Oil Cryptography.

14375C20F07A9DF2EBDE78F063C5AD7C

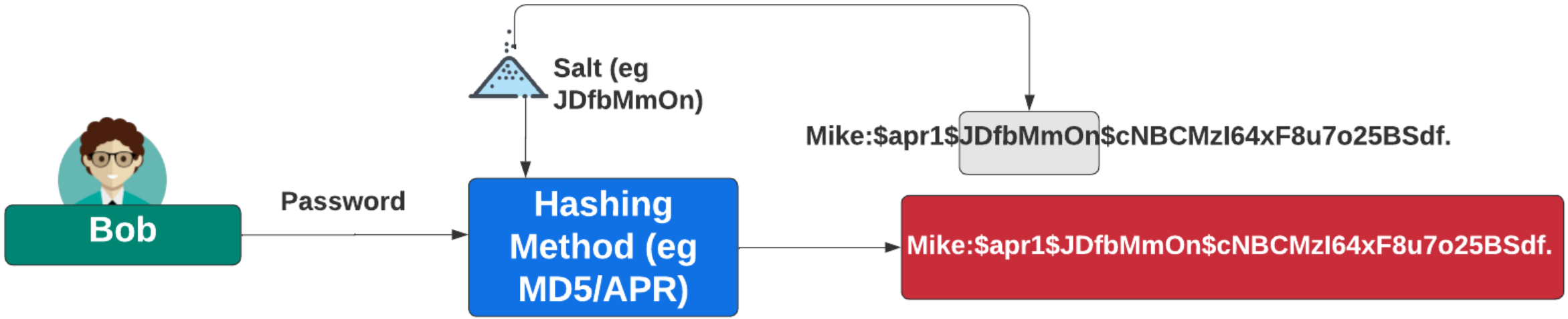
FDdcIPB6nfLr3njwY8WtfA==

There are two kinds of cryptography in this world: cryptography that will stop your kid sister from reading your files, and cryptography that will **top** major governments from reading your files. Cryptography is typically bypassed, not penetrated. Security is a process, not a product. Beware of Snake Oil Cryptography.

```
Linux command: echo -n "Hello" | openssl dgst -md5
Windows command: echo | set /p = "Hello" | openssl dgst -md5

Message: Hello
Mode: md5
========
8b1a9953c4611296a827abf8c47804d7
```

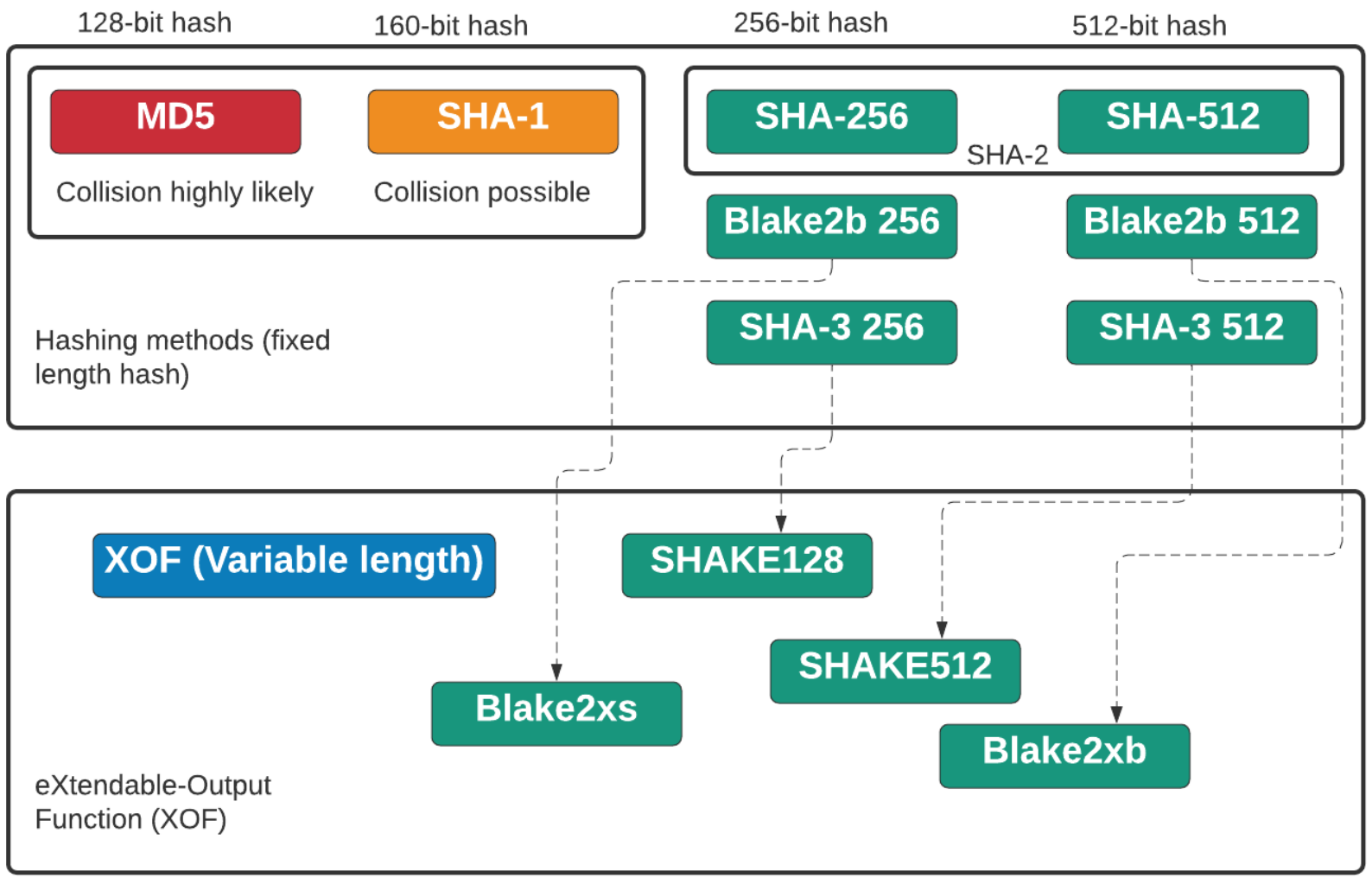
https://asecuritysite.com/openssl/openssl_full2b
https://asecuritysite.com/openssl/openssl_full2

Adding Salt



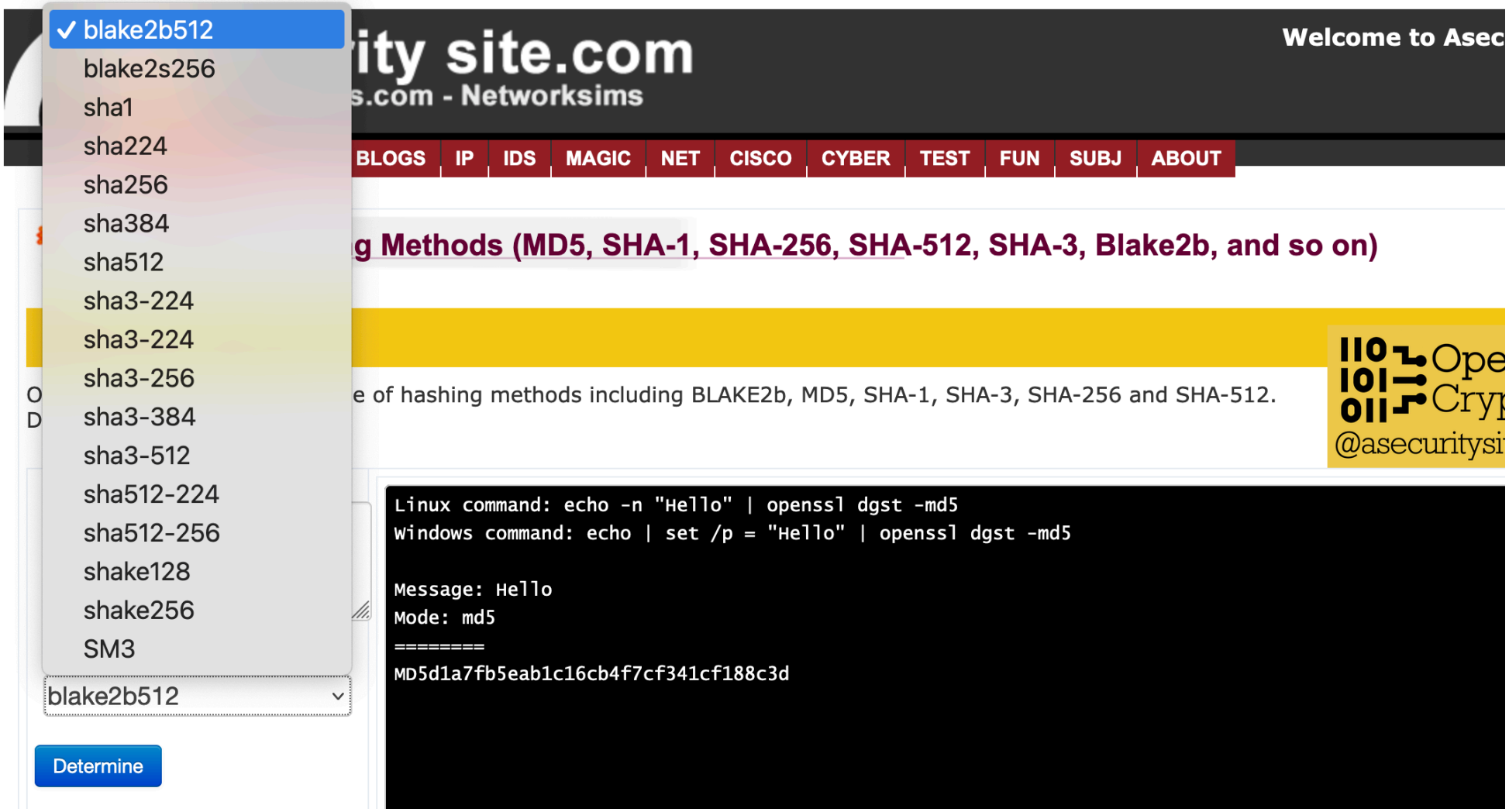
https://asecuritysite.com/hash/salt

MD5, SHA-1, SHA-2, SHA-3 and XOF



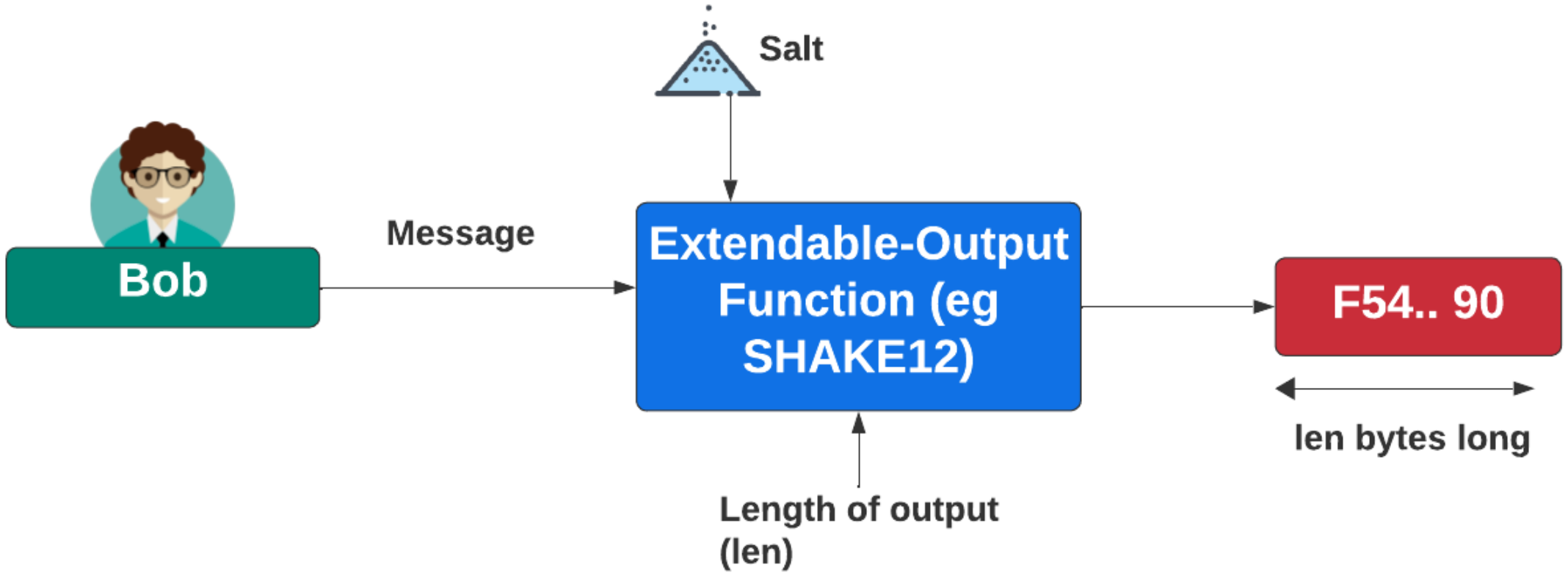
https://asecuritysite.com/hash/cs

OpenSSL



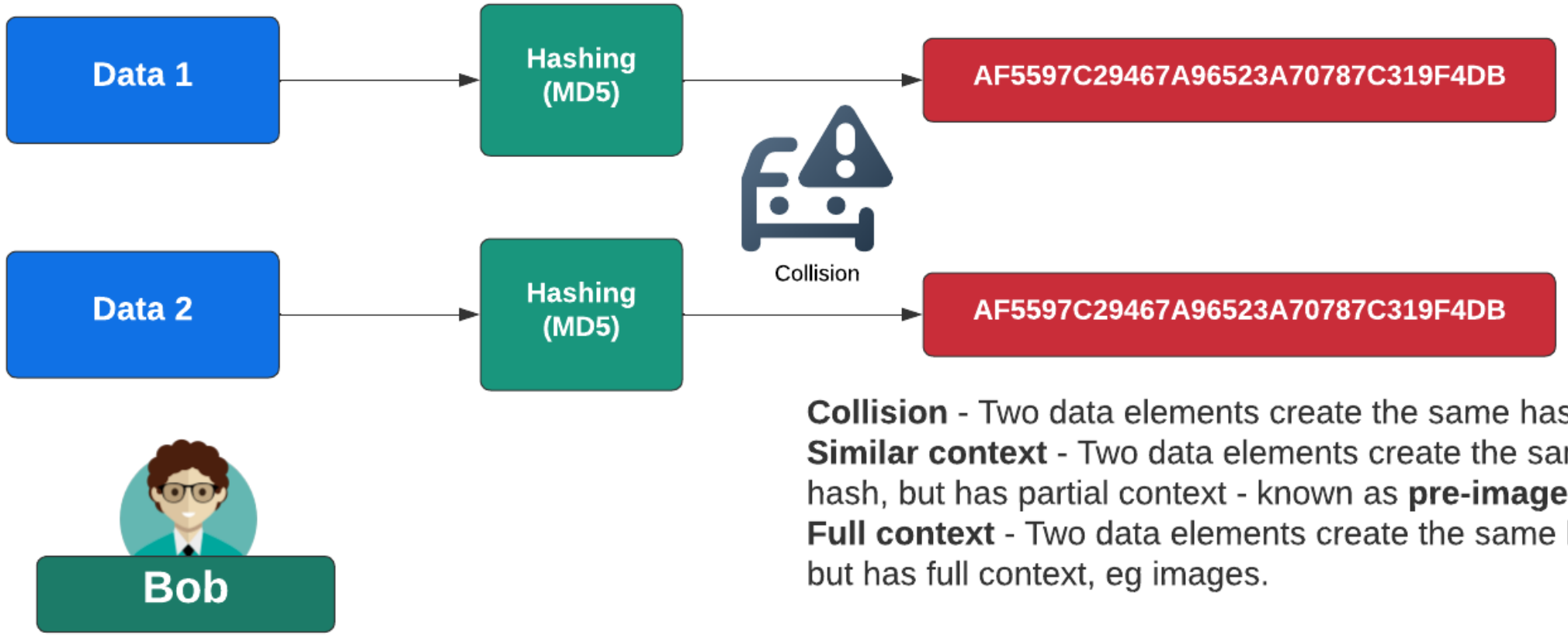
https://asecuritysite.com/openssl/openssl_full2

XOF



https://asecuritysite.com/hash/cs

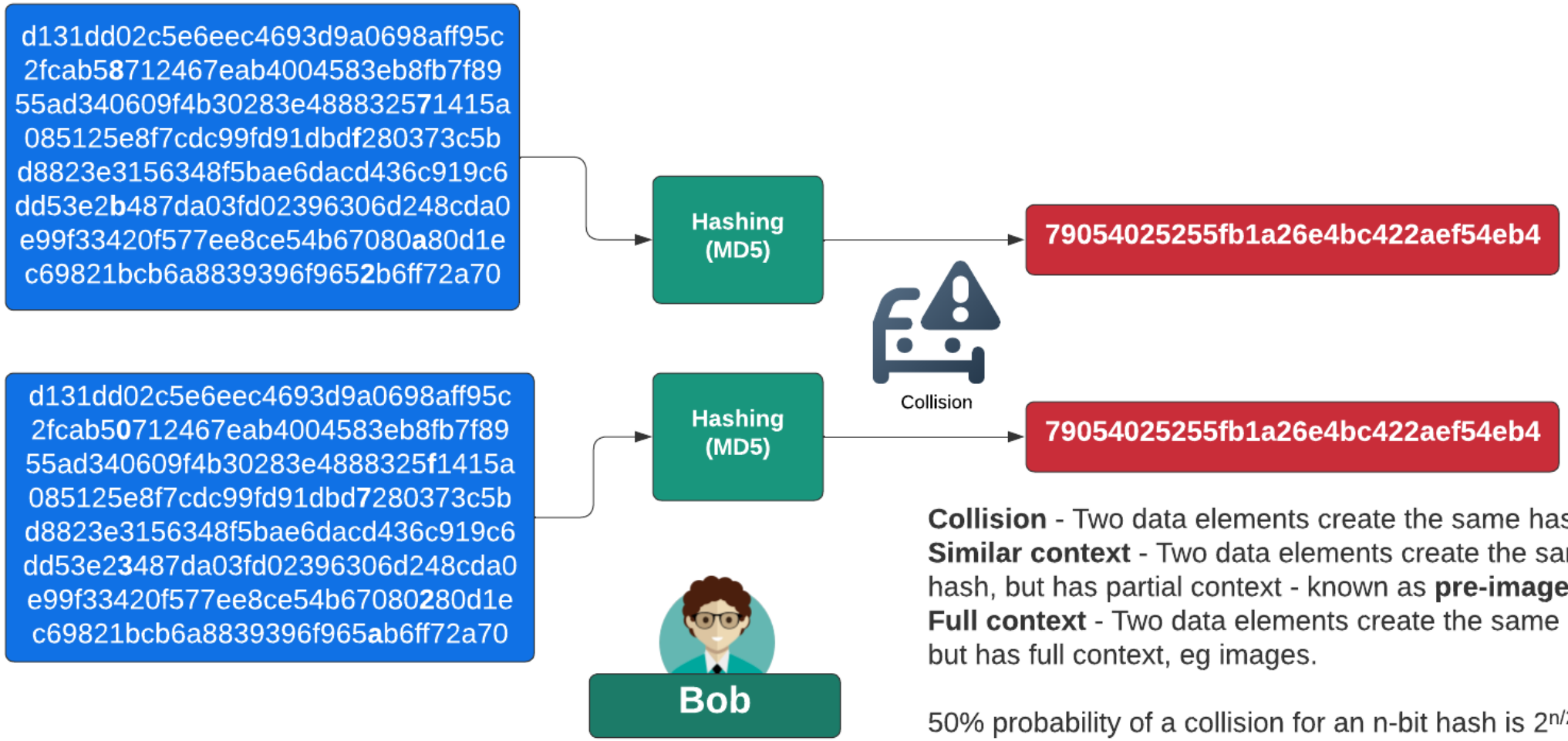
Hash Collisions



Collision - Two data elements create the same hash
Similar context - Two data elements create the same hash, but has partial context - known as **pre-image attack**.
Full context - Two data elements create the same hash, but has full context, eg images.

50% probability of a collision for an n-bit hash is $2^{n/2}$

A Real Hash Collision Due To A Flaw



Collision - Two data elements create the same hash
Similar context - Two data elements create the same hash, but has partial context - known as **pre-image attack**.
Full context - Two data elements create the same hash, but has full context, eg images.

50% probability of a collision for an n-bit hash is $2^{n/2}$

Hash Collision Using Birthday Attack

Hashing (MD5)

e06723d4961a0a3f950e7786f3766338

Collision

Hashing (MD5)

e06723d4961a0a3f950e7786f3766338

Bob

In 2014, Nat McHugh created a hash collision on two images

- 10 hours of computing on the Amazon GPU Cloud.
- Cost: 60 cents
- Used: Hashcat (on CUDA)
- Birthday attack: A group size of only 70 people results in a 99.9% chance of two people sharing the same birthday.
- M-bit output there are 2^m messages, and the same hash value would only require $2^{m/2}$ random messages.

Hash Benchmarks

```
Ulta fast:
Murmur:          545,716 hashes per second

Fast:
SHA-1:           134,412
SHA-256:         126,323
MD5:             125,741
SHA-512:          76,005
SHA-3 (224-bit): 72,089

Medium speed:
LDAP (SHA1):     13,718
MS DCC:           9,582
NT Hash:          7,782
MySQL:            7,724
Postgres (MD5):   7,284

Slow:
PBKDF2 (SHA-256): 5,026
Cisco PIX:        4,402
MS SQL 2000:      4,225
LDAP (MD5):       4,180
Cisco Type 7:     3,775
PBKDF2 (SHA1):    2,348

Ulta-slow:
LM Hash:            733
APR1:               234
Bcrypt:             103
DES:                 88
Oracle 10:           48
```

Hashes “The quick brown fox jumps over the lazy dog:

SHA-1: 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
SHA-256: d7a8fbb307d7809469ca9abcb0082e4f8d5651e46d3cdb762d02d0bf37c9e592
SHA-512: 07e547d9586f6a73f73fbac0435ed76951218fb7d0c8d788a309d7
85436bbb642e93a252a954f23912547d1e8a3b5ed6e1bfd7097821233fa0538f3db854fee6

MD-5: 9e107d9d372bb6826bd81d3542a419d6
DES: ZDeS94Lcq/6zg
Bcrypt: $2a$05$2czCv5GYgkx3aobmEyewB.ejV2hePMdbvTdCyNaSzWtlGPPjB2xx6
APR1: $apr1$ZDzPE45C$3PvRanPycmNc6c2G9wT9b/
PBKDF2 (SHA1): $pbkdf2$5$WkR6UEU0NUM$0RB2bimWrMY.EPYibpaBT2q3HFg
PBKDF2 (SHA-256): $pbkdf2-sha256$5$WkR6UEU0NUM$yrJz2oJix7uBJZwZ/50vWUgdE
I/i0ffqeU4obqC0pk4
LM Hash: a7b07f9948d8cc7f97c4b0b30cae500f
NT Hash: 4e6a076ae1b04a815fa6332f69e2e231
MS DCC: efa9778bbc94a7360f664eb7d7144725
LDAP (MD5): {MD5}9e107d9d372bb6826bd81d3542a419d6
LDAP (SHA1): {SHA}2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
MS SQL 2000: 0x0100BF77CE595DCD1FC87A37B3DEBC27A8C97355CB96B8BAB
63E602662BA5D5D33B913E422499BE72FF3D9BB65DE
MySQL: *A4E4D26FD0C6455E23E2187C3AABE844332AA1B3
Oracle 10: 4CDA2299FCAD0499
Postgres (MD5): md5d44c15daa11770f25c5350f7e5408dd1
Cisco PIX: kGyKN5CqdFQ1qJUs
Cisco Type 7: 15260309443B3E2D2B3875200108010D41505640135E1B0E080
5195741564015400 35E460B594D1D53020B5C

Benchmark

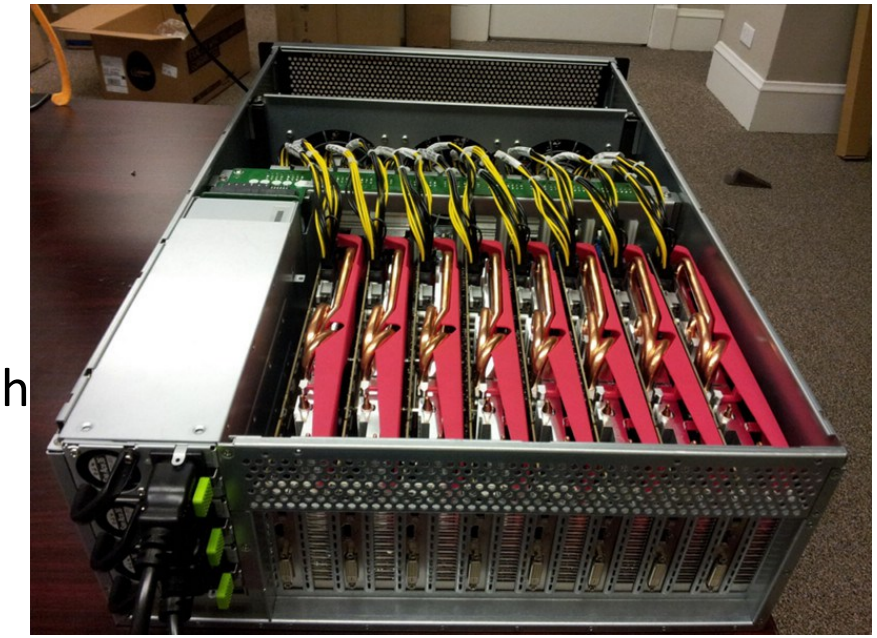
Hash Crackers/Bit Coin Miners

Fast Hash One

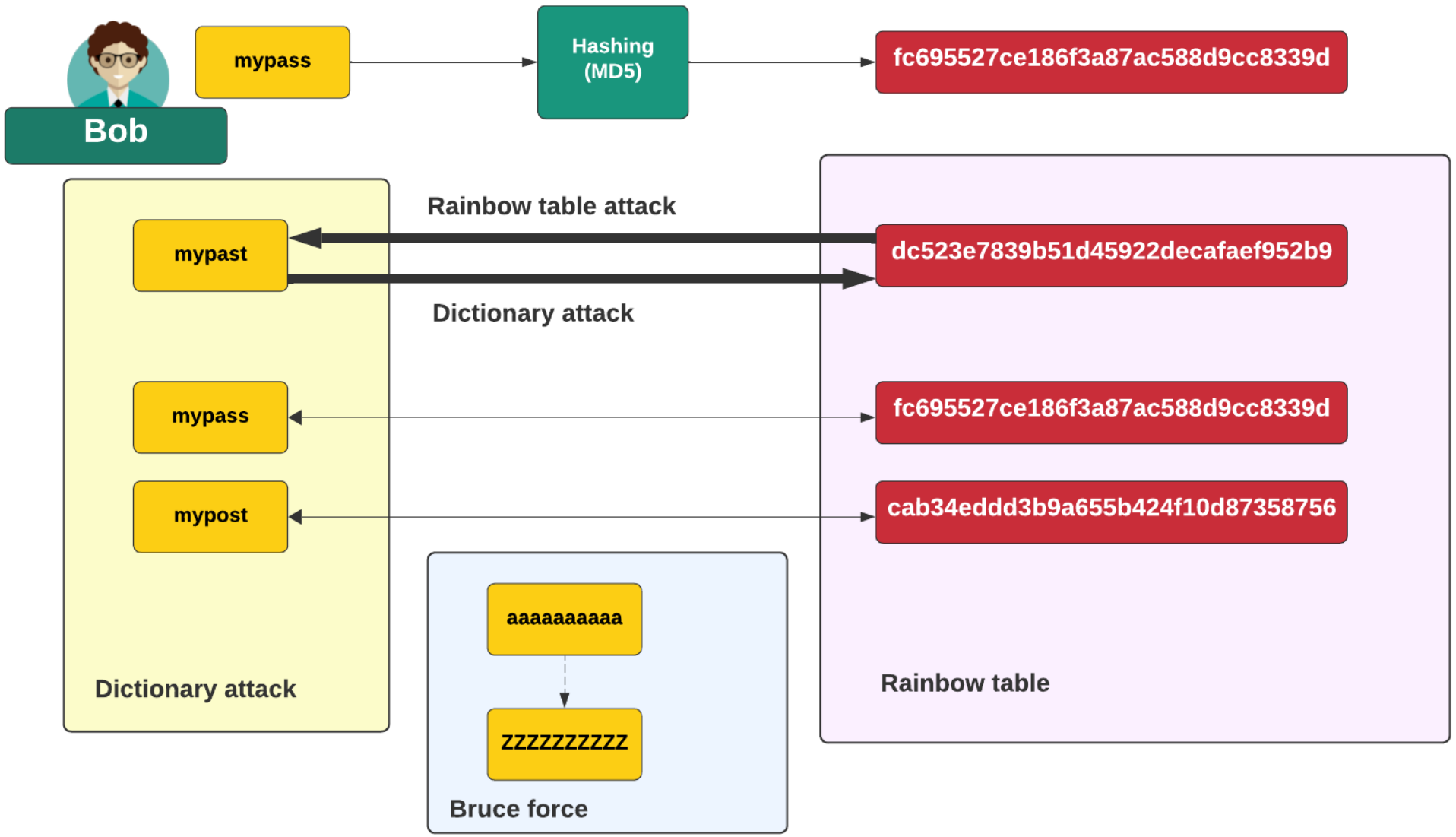
- 1.536TH/s – Cost 3-5,000 dollars.

25 GPU Hash Cracker

- An eight character NTLM password cracked in 5.5 hours. 14 character LM hash cracked in six minutes. 350 billion hashes per second.



Cracking Methods



Brute Force - How many hash codes?

- 7 digit password with [a-z] … how many?
  - Ans:
  - Time to crack - 100 billion per second:
- 7 digit with [a-zA-z] … how many?
  - Ans:
  - Time to crack – 100 billion per second:
- 7 digit with [a-zA-z!@#$%^&*()] … how many?
  - Ans:
  - Time to crack – 100 billion per second:

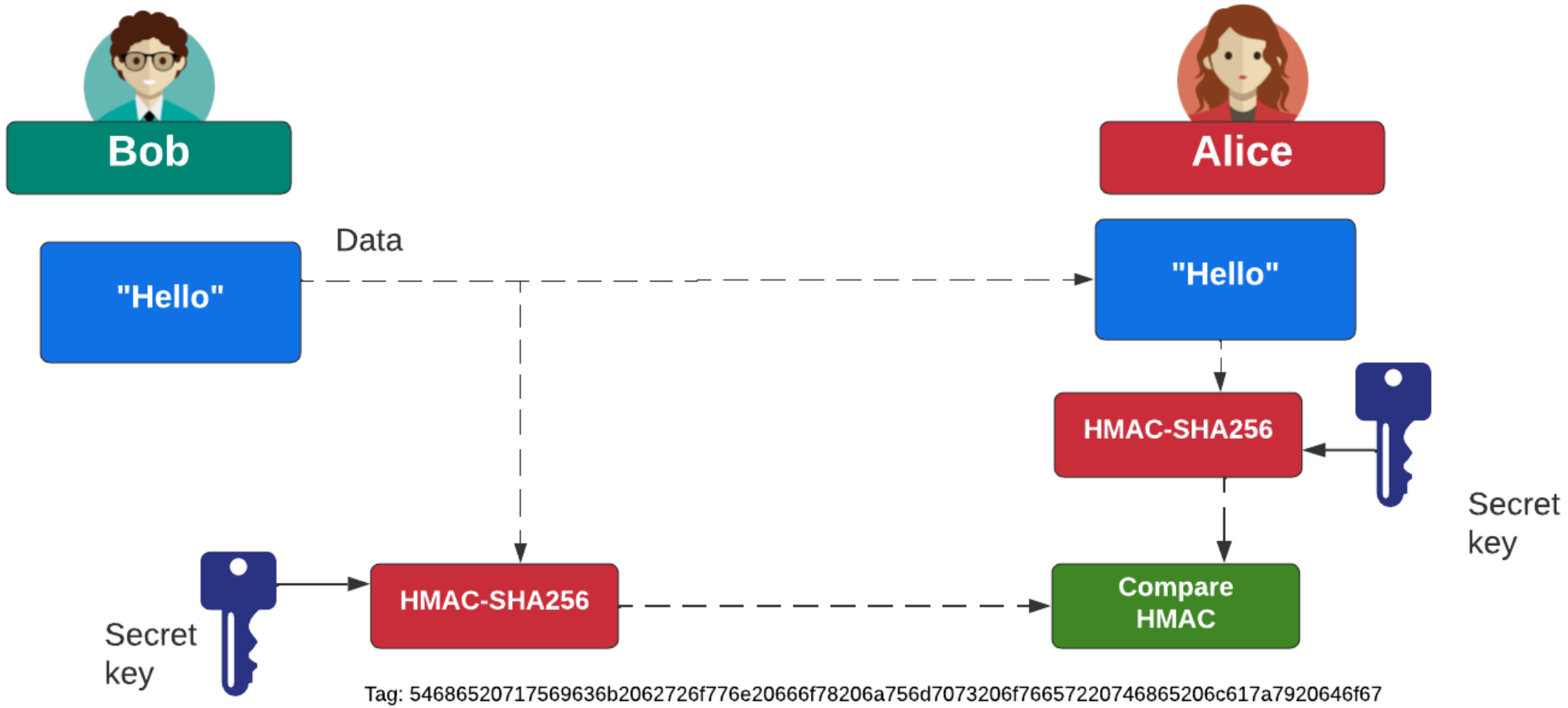
Cryptography: MAC

Prof Bill Buchanan OBE FRSE

https://asecuritysite.com/mac

Fundamentals
Symmetric Key
Hashing
MAC
KDF
Public Key
Key Exchange
Signatures
Digital Certificates

HMAC (Key Hash Message Authentication Code)



https://asecuritysite.com/mac/hashnew2

HMAC (Key Hash Message Authentication Code)

Some AES modes are vulnerable to bit flipping attacks. In this case, we will use AES CTR mode, and flip the bits of the nth character in the ciphertext, and where we specify the character to target (by default, the program will focus on changing the 9th character).

Data:

Pay Bob 1 Dollar

Character to flip:

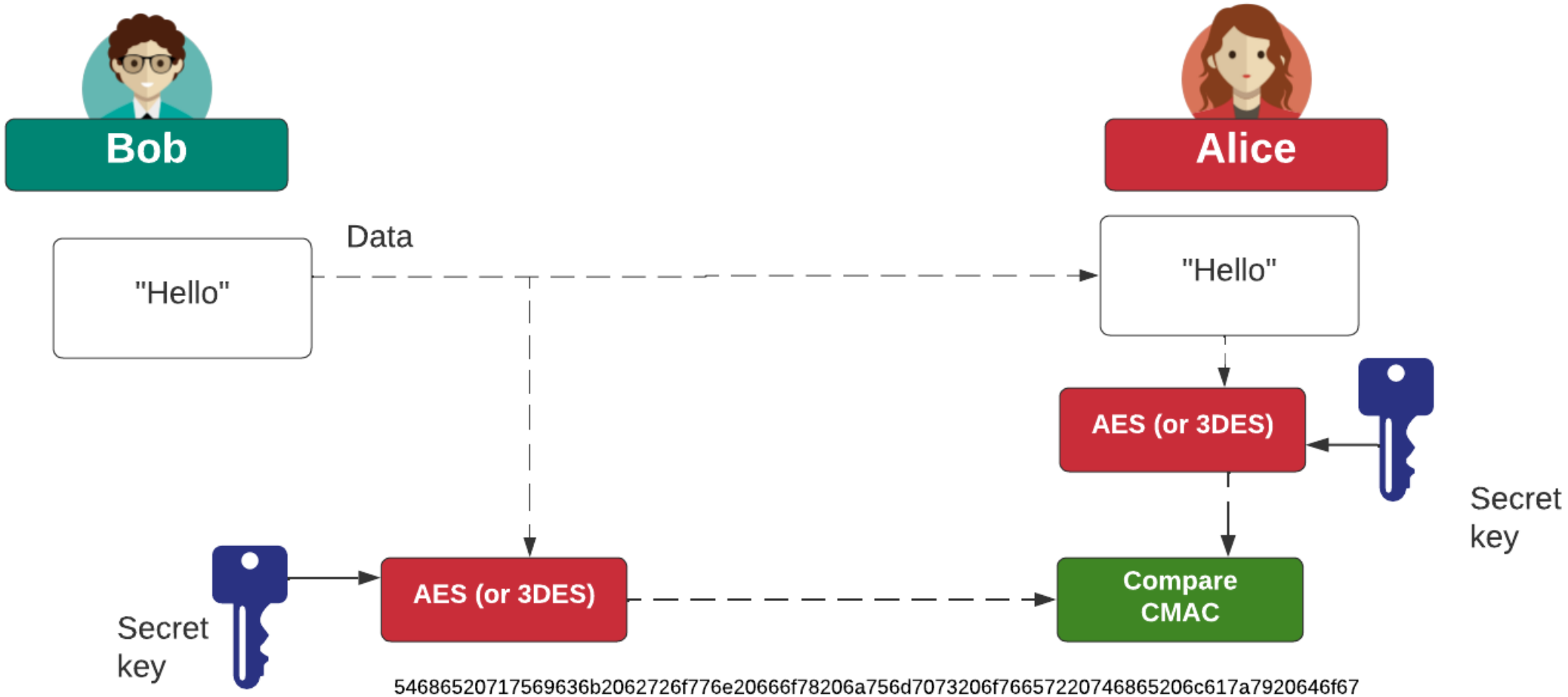
1st

Determine

```
Message:          Pay Bob 1 Dollar
Key:      b'333dfcec027543d99aba11bf74262e6763ab65095a0236494a2ef5d527a49b7b'
IV:       b'7c7833650705f323ee236e33844889e4'


=== AES CTR ===
Flipping two least significant bits in character 9

Before bitflip: 76368b817f252424d668369378fdae0b
After bitflip: 76368b817f252424d568369378fdae0b

Cipher:   b'76368b817f252424d568369378fdae0b'
```

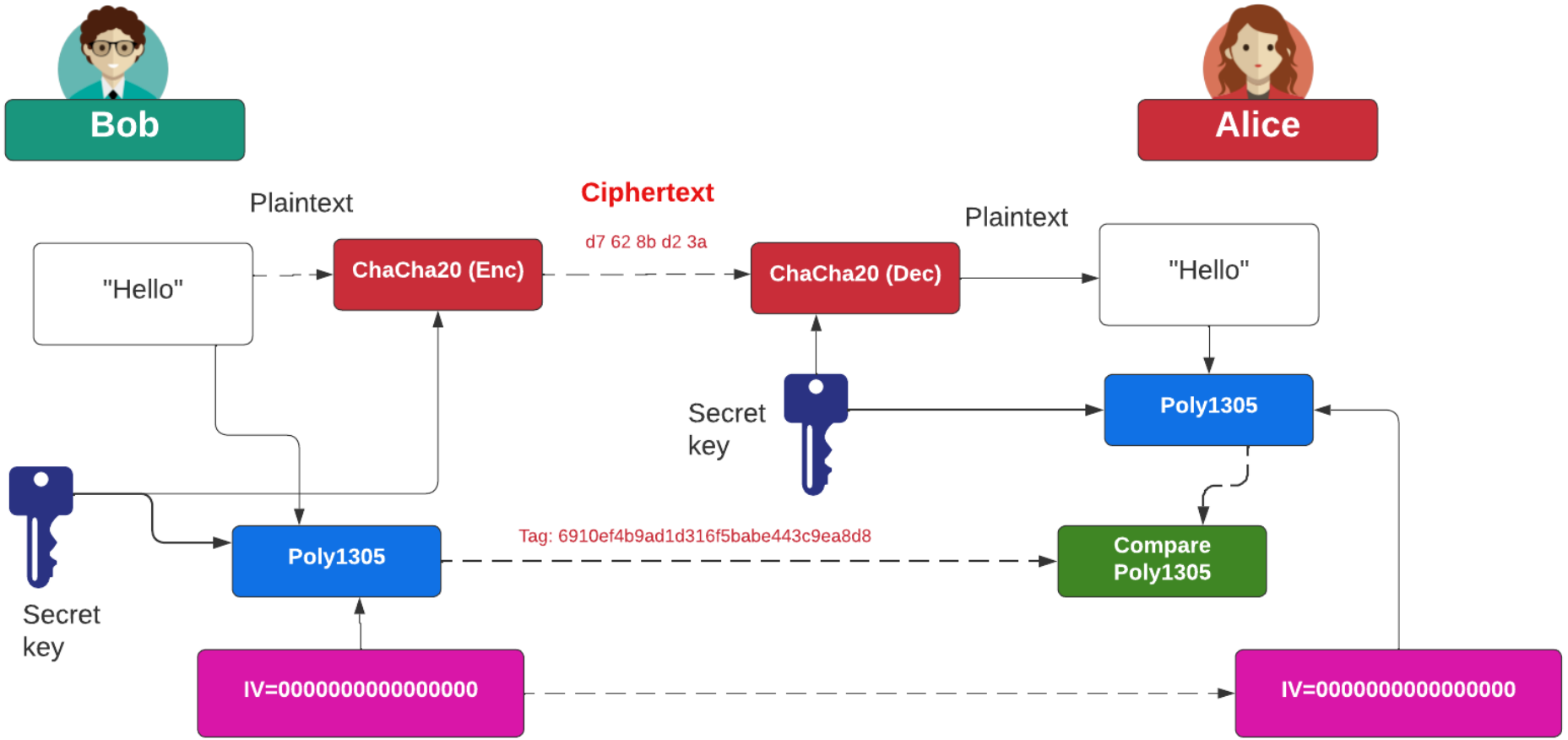
https://asecuritysite.com/aes/hashnew4a

CMAC (Cipher MAC)



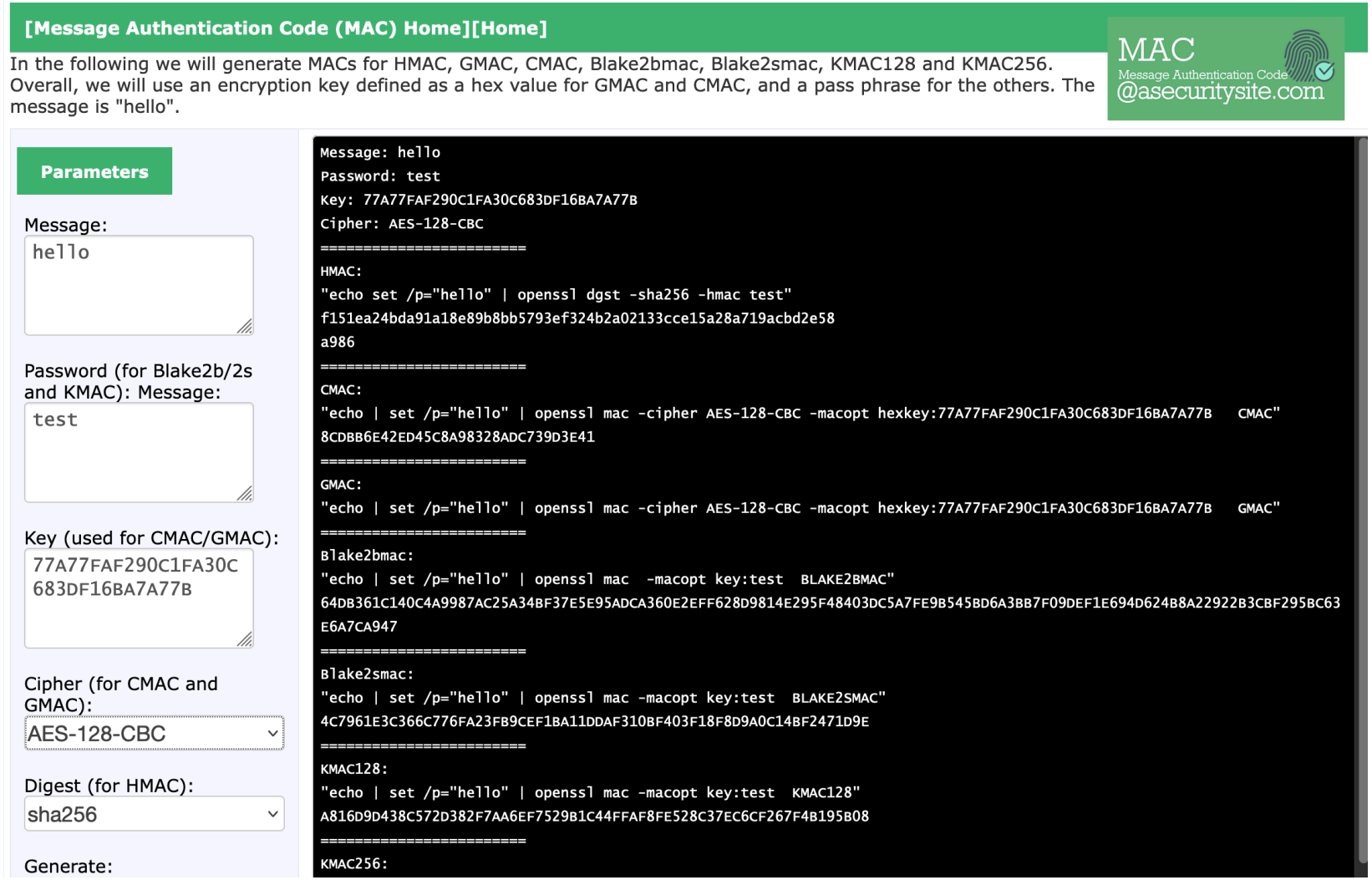
https://asecuritysite.com/golang/go_hmac

ChaCha20/Poly1305



https://asecuritysite.com/golang/go_hmac

Generating MACs with OpenSSL



https://asecuritysite.com/mac/openssl_mac

Cryptography: KDF

Prof Bill Buchanan OBE FRSE

https://asecuritysite.com/kdf

Fundamentals
Symmetric Key
Hashing
MAC
KDF
Public Key
Key Exchange
Signatures
Digital Certificates

BLOCKPASS IDENTITY LAB
World-leading Collaboration between
Blockpass IDN and Edinburgh Napier University

Keys Generated By Passwords With Brute Force

Characters in password:
Characters [a-z]: ☑
Characters [A-Z]: ☑
Characters [0-9]: ☐
Characters [!@#$%^&*()+_]: ☐
Characters [~`-_=+[]{}\|;:\'",.<>?/]: ☐

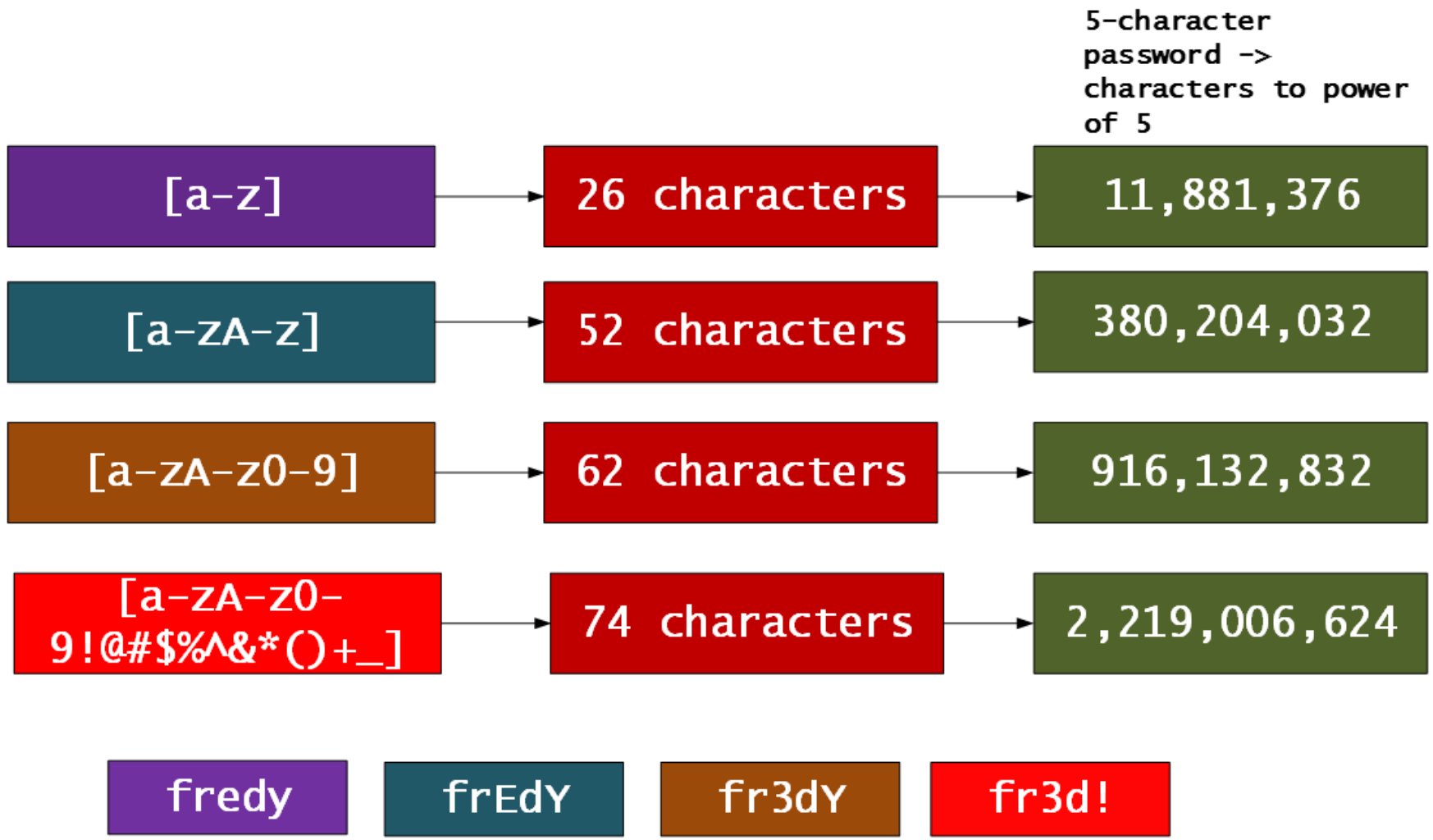
Password cracking speed: 1 Million per second **Processing elements:** 1

Results

No of characters:	52	Time to crack (max)	Key entropy (bits)
No of passwords (5 digits):	380,204,032	6.34 mins	28.5
No of passwords (6 digits):	19,770,609,664	5.49 hours	34.2
No of passwords (7 digits):	1,028,071,702,528	11.90 days	39.9
No of passwords (8 digits):	53,459,728,531,456	1.69 years	45.6
No of passwords (9 digits):	2,779,905,883,635,712	88.09 years	51.3
No of passwords (10 digits):	144,555,105,949,057,024	4580.78 years	57
No of passwords (11 digits):	7,516,865,509,350,965,248	238200.38 years	62.7
No of passwords (12 digits):	390,877,006,486,250,192,896	12386419.66 years	68.41

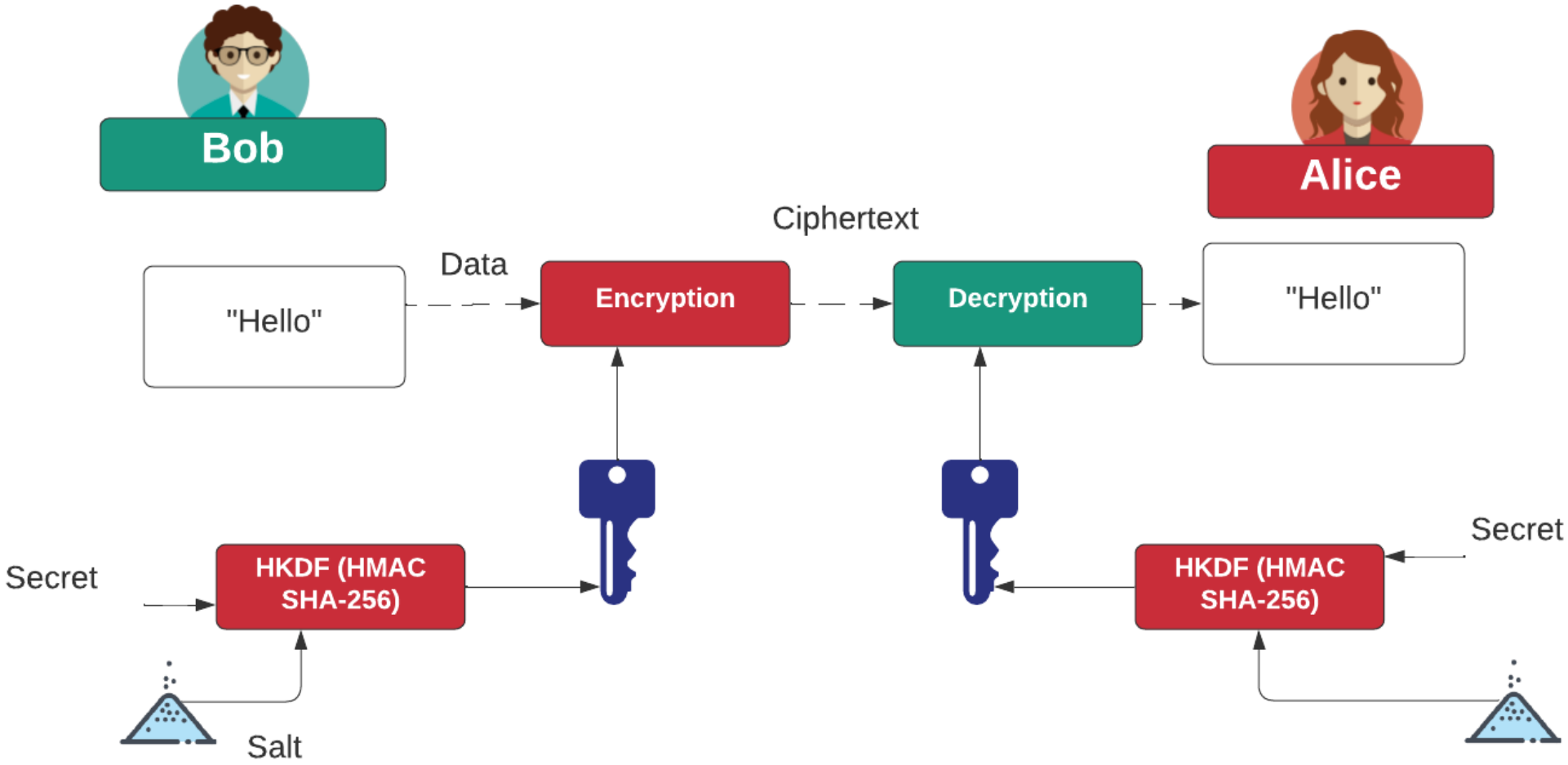
https://asecuritysite.com/encryption/passes

Keys Generated By Passwords With Brute Force



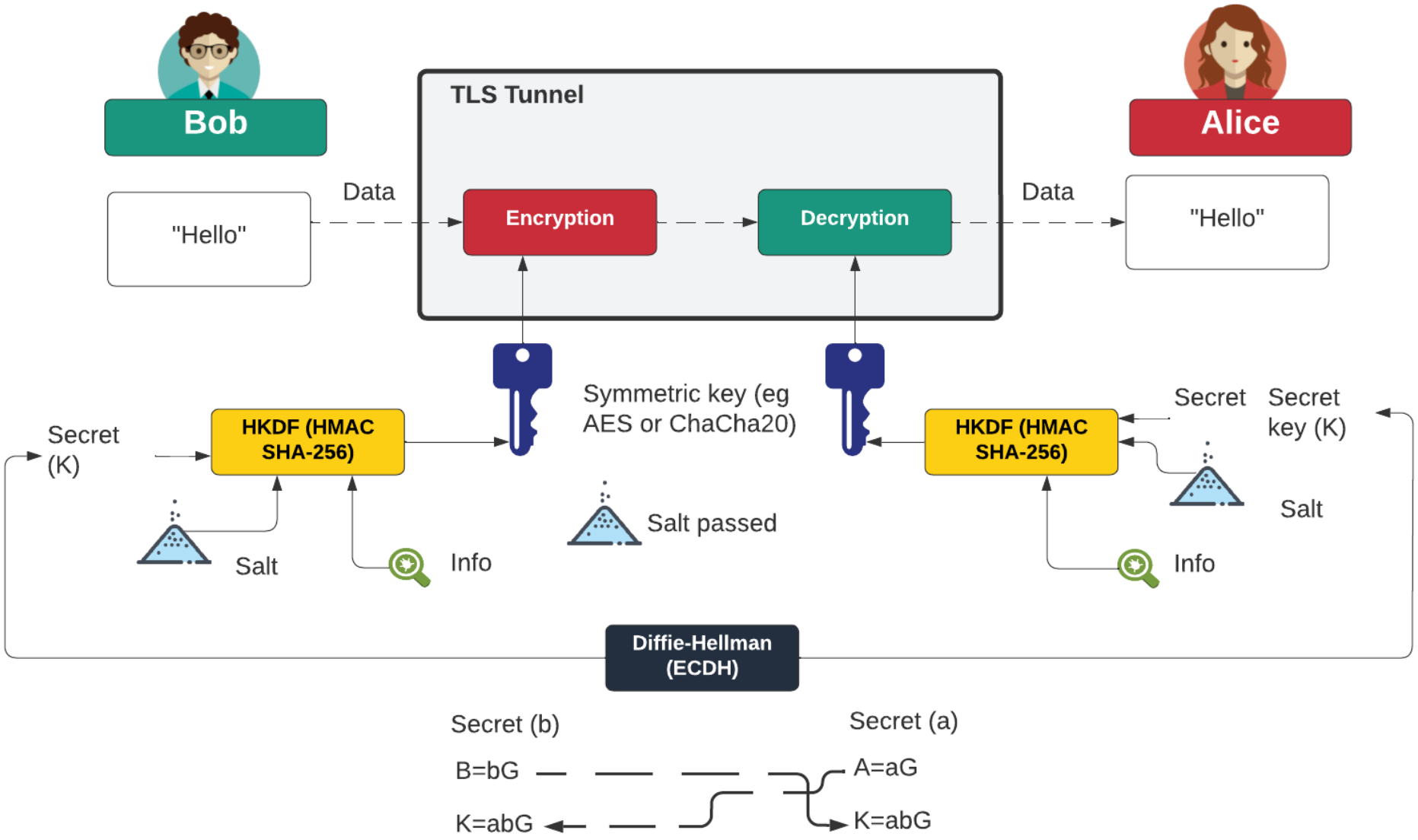
https://asecuritysite.com/encryption/passes

HKDF (HMAC Key Derivation Function)

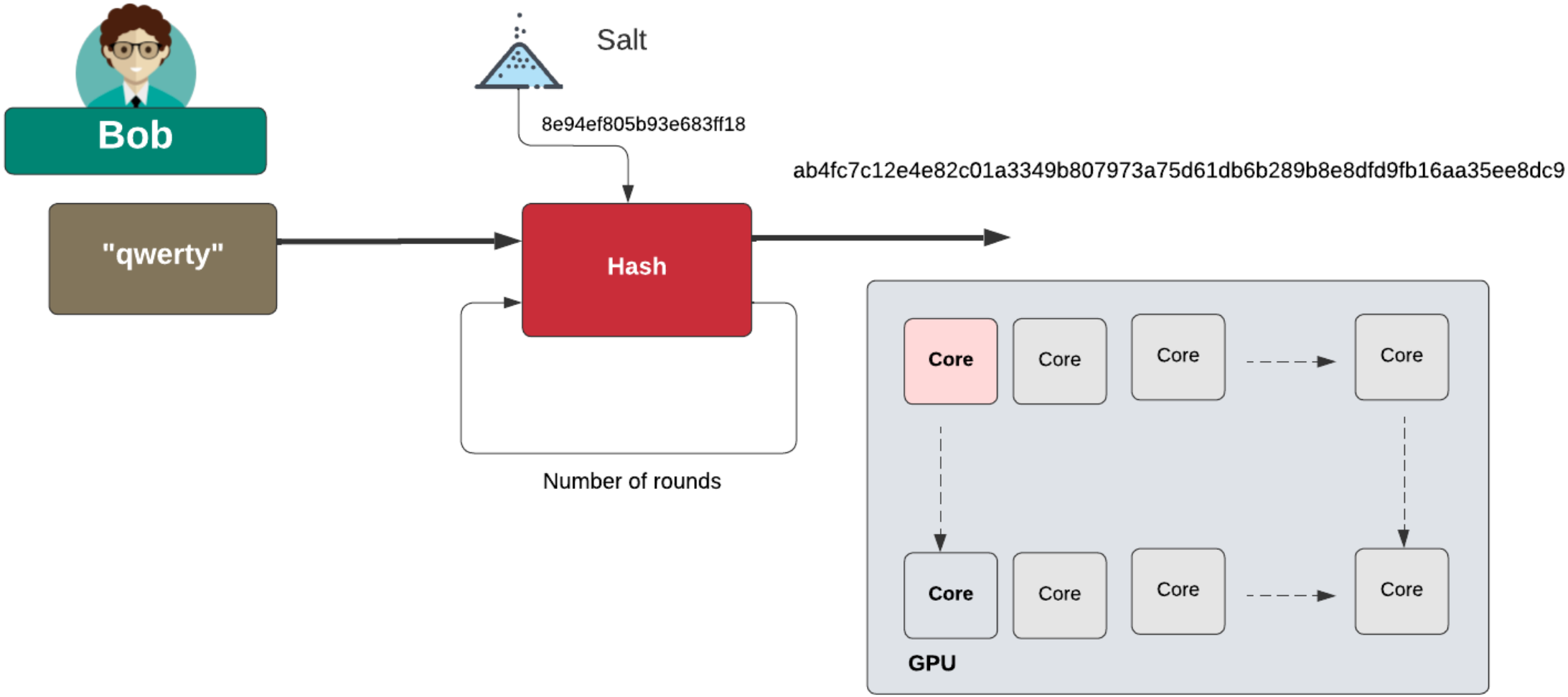


https://asecuritysite.com/pbkdf2/hashnew3

TLS Setup of Symmetric Key

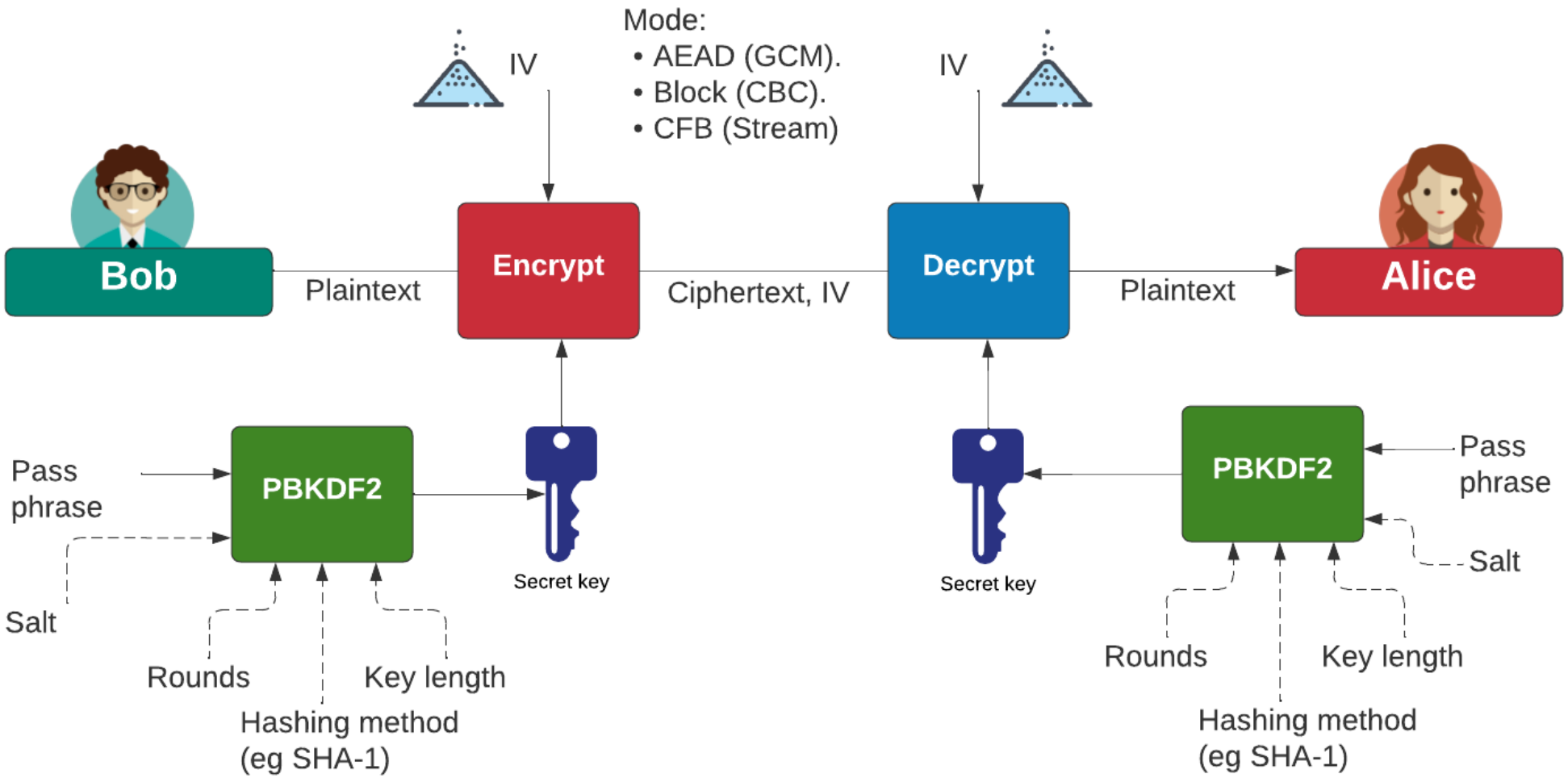


Slowing down hashing



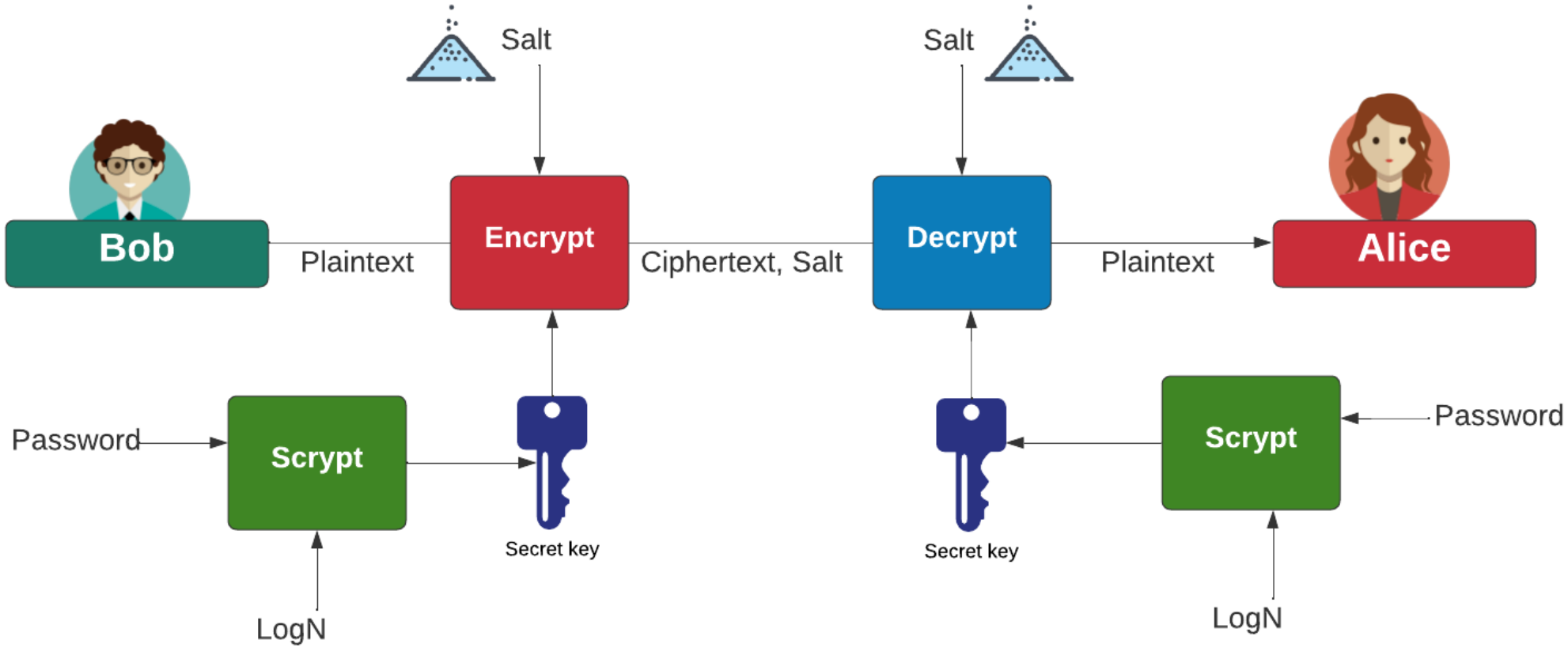
https://asecuritysite.com/hash/htest

PBKDF2



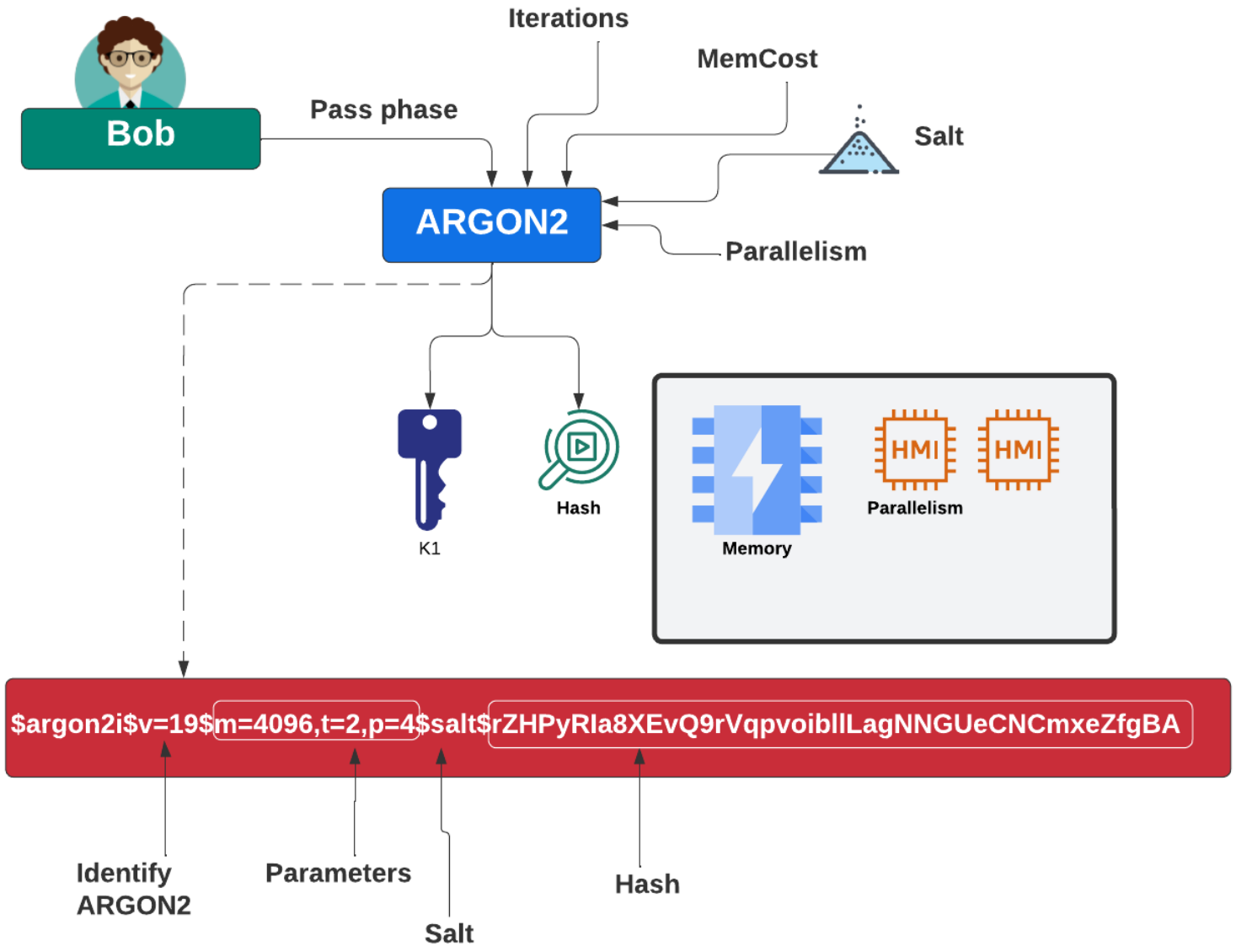
https://asecuritysite.com/pbkdf2/PBKDF2_2

scrypt



https://asecuritysite.com/pbkdf2/kdf

Argon2



https://asecuritysite.com/encryption/argon

Argon2

Argon2				Argon2				Argon2			
Cost			Time	Cost			Time	Cost			Time
n	log_2(m)	p	sec.	n	log_2(m)	p	sec.	n	log_2(m)	p	sec.
1	8	1	0.003	1	8	2	0.002	1	8	4	0.002
2	8	1	0.003	2	8	2	0.003	2	8	4	0.002
4	8	1	0.005	4	8	2	0.004	4	8	4	0.002
8	8	1	0.008	8	8	2	0.006	8	8	4	0.003
16	8	1	0.015	16	8	2	0.011	16	8	4	0.005
32	8	1	0.029	32	8	2	0.016	32	8	4	0.008
64	8	1	0.051	64	8	2	0.027	64	8	4	0.015
128	8	1	0.105	128	8	2	0.044	128	8	4	0.029
256	8	1	0.151	256	8	2	0.079	256	8	4	0.057
512	8	1	0.362	512	8	2	0.143	512	8	4	0.117
1024	8	1	0.631	1024	8	2	0.327	1024	8	4	0.237
2048	8	1	1.397	2048	8	2	0.595	2048	8	4	0.474
4096	8	1	2.721	4096	8	2	1.291	4096	8	4	0.946
8192	8	1	5.787	8192	8	2	2.395	8192	8	4	1.918
16384	8	1	11.575	16384	8	2	4.859	16384	8	4	3.777
32768	8	1	24.095	32768	8	2	9.796	32768	8	4	7.591

The parameters include:

- Password (P): Defines the password bytes to be hashed
- Salt (S): Defines the bytes to be used for salting.
- Parallelism (p): Defines the number of thread that are required for the parallelism.
- TagLength (T): Define the number of bytes to return for the hash.
- MemorySizeKB (m): Amount of memory (in KB) to use.

A sample run is:

Message: abc Hash: $argon2i$v=19$m=8,t=1,p=1$x/3yHil0MIE$RDL1Jw

https://asecuritysite.com/encryption/argon

OWASP

Use Argon2id with a minimum configuration of 19 MiB of memory, an iteration count of 2, and 1 degree of parallelism.

If Argon2id is not available, use scrypt with a minimum CPU/memory cost parameter of (2^17), a minimum block size of 8 (1024 bytes), and a parallelization parameter of 1.

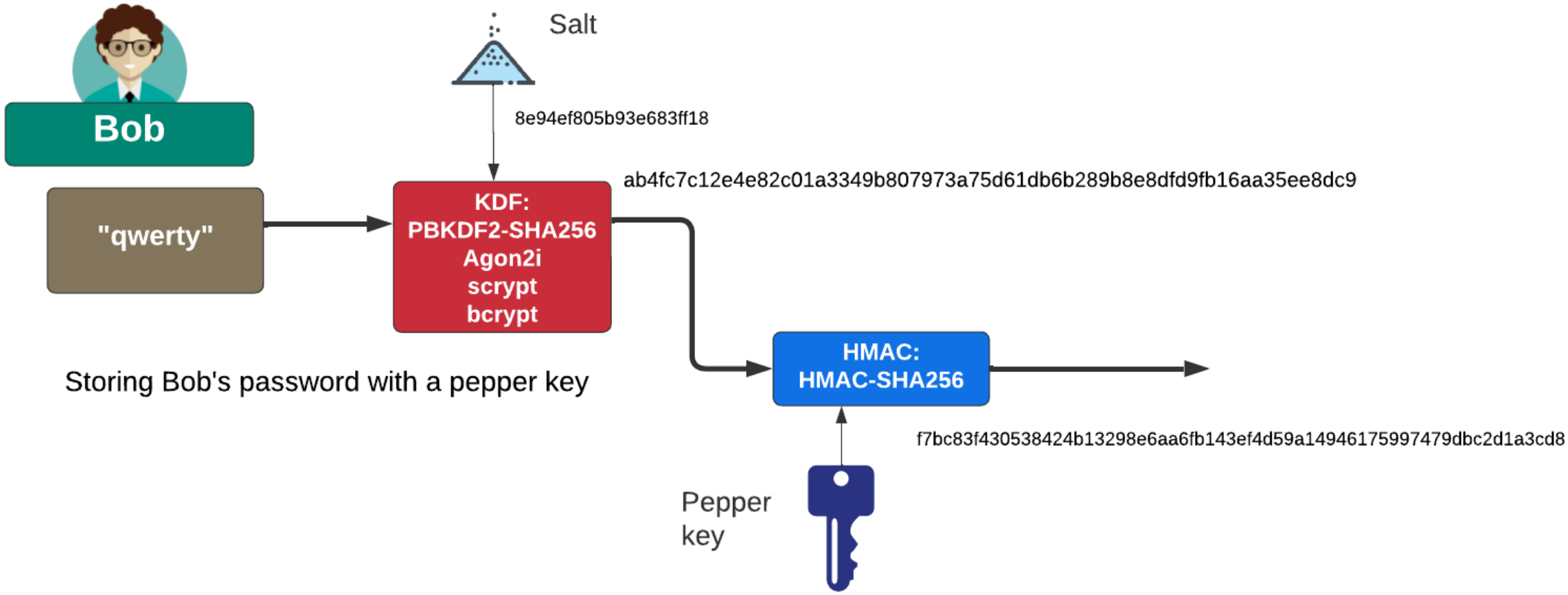
For legacy systems using bcrypt, use a work factor of 10 or more and with a password limit of 72 bytes.

If FIPS-140 compliance is required, use PBKDF2 with a work factor of 600,000 or more and set with an internal hash function of HMAC-SHA-256.

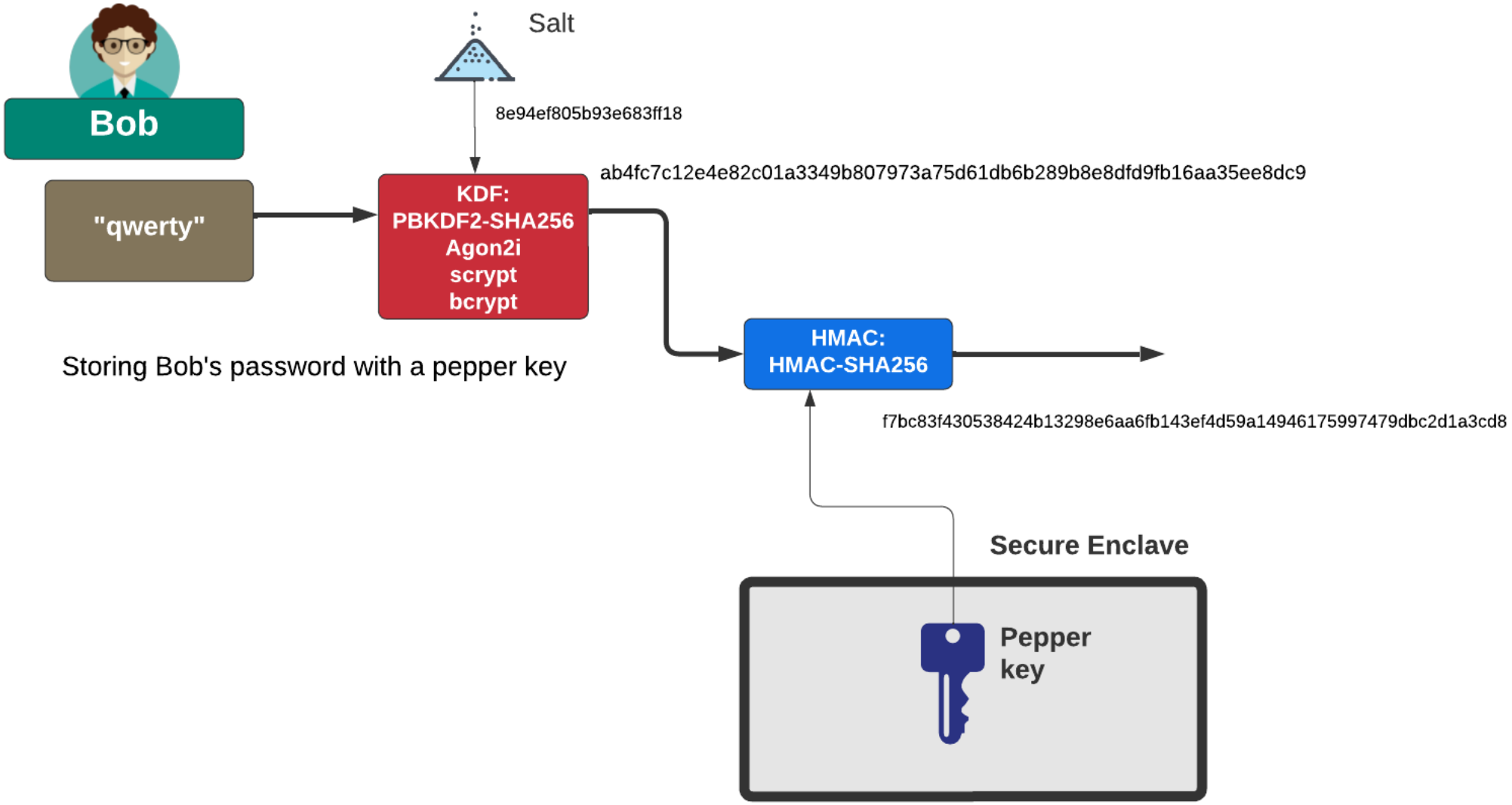
Consider using a pepper to provide additional defense in depth (though alone, it provides no additional secure characteristics).

https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

Using a Pepper Key



Using a Pepper Key



Storing Bob's password with a pepper key

Cryptography: Public Key

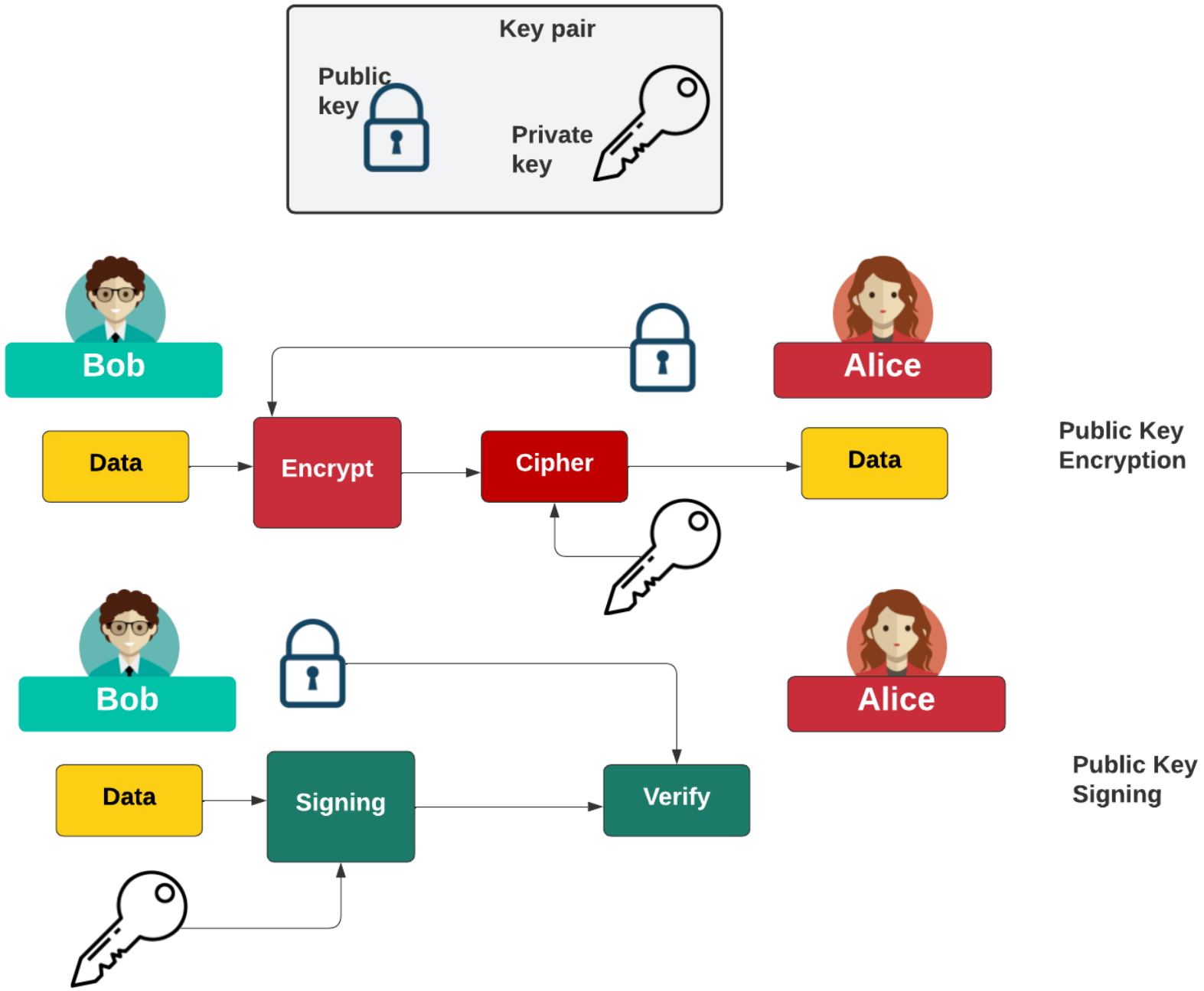
Prof Bill Buchanan OBE FRSE

- Fundamentals
- Symmetric Key
- Hashing
- MAC
- KDF
- **Public Key**
- Key Exchange
- Signatures
- Digital Certificates

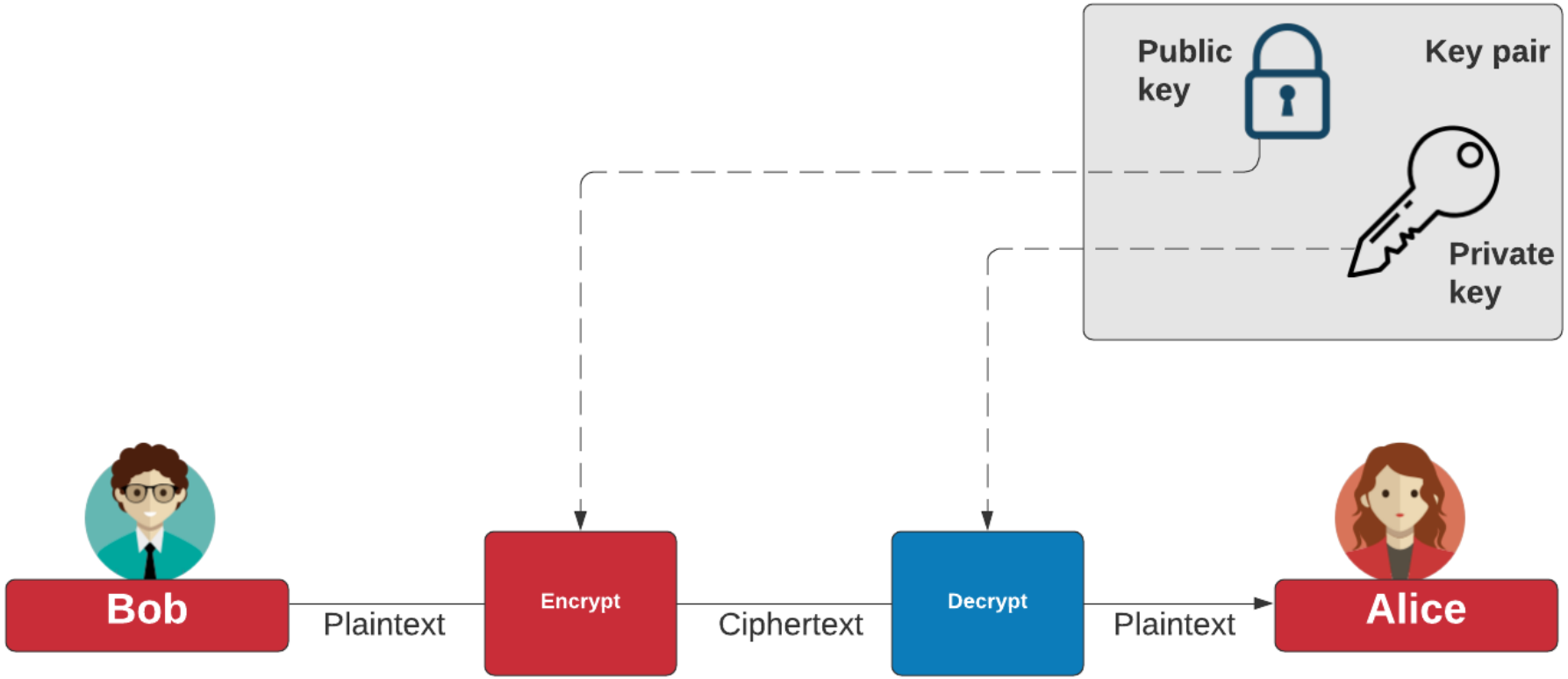
https://asecuritysite.com/rsa

https://asecuritysite.com/ecc

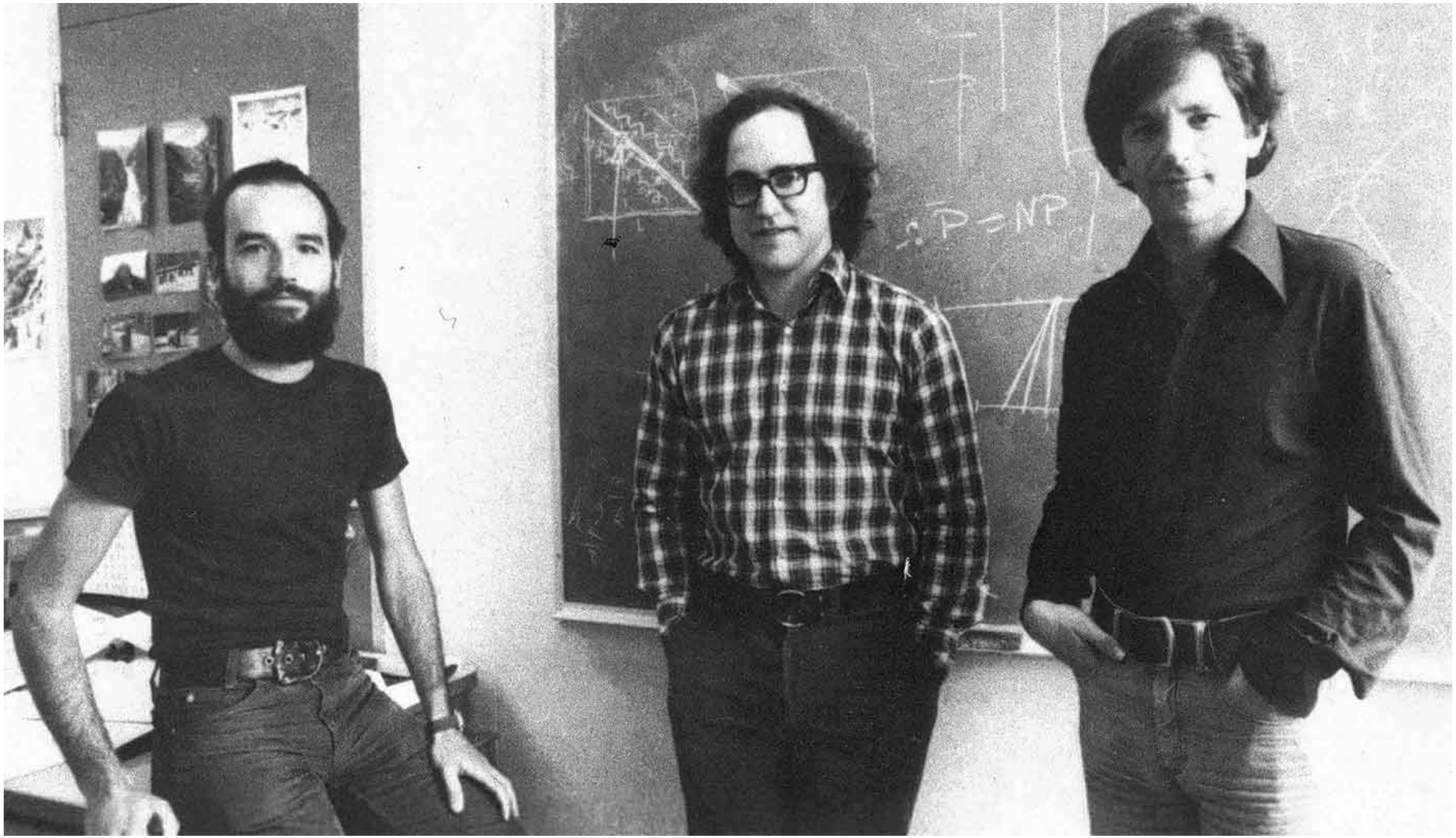
Public Key


Key pair
Public key
Private key
Bob
Alice
Data
Encrypt
Cipher
Data
Public Key Encryption
Bob
Alice
Data
Signing
Verify
Public Key Signing

Public Key


Public key
Key pair
Private key
Bob
Plaintext
Encrypt
Ciphertext
Decrypt
Plaintext
Alice

Public Key



Public Key

LABORATORY FOR COMPUTER SCIENCE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MIT/LCS/TM-82

A METHOD FOR OBTAINING DIGITAL SIGNATURES AND PUBLIC-KEY CRYPTOSYSTEMS

Ronald Rivest
Adi Shamir
Len Adleman

April 1977

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Programming Techniques — S.L. Graham, R.L. Rivest*, Editors

A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

R. L. Rivest, A. Shamir, and L. Adleman
MIT Laboratory for Computer Science
and Department of Mathematics

An encryption method is presented with the novel property that publicly revealing an encryption key does not thereby reveal the corresponding decryption key. This has two important consequences: (1) Couriers or other secure means are not needed to transmit keys, since a message can be enciphered using an encryption key publicly revealed by the intended recipient. Only he can decipher the message, since only he knows the corresponding decryption key. (2) A message can be "signed" using a privately held decryption key. Anyone can verify this signature using the corresponding publicly revealed encryption key. Signatures cannot be forged, and a signer cannot later deny the validity of his signature. This has obvious applications in "electronic mail" and "electronic funds transfer" systems. A message is encrypted by representing it as a number M, raising M to a publicly specified power e, and then taking the remainder when the result is divided by the publicly specified product, n, of two large secret prime numbers p and q. Decryption is similar; only a different, secret, power d is used, where $e \cdot d \equiv 1 \pmod{(p-1)\cdot(q-1)}$. The security of the system rests in part on the difficulty of factoring the published divisor, n.

Key Words and Phrases: digital signatures, public-key cryptosystems, privacy, authentication, security, factorization, prime number, electronic mail, message-passing, electronic funds transfer, cryptography.
CR Categories: 2.12, 3.15, 3.50, 3.81, 5.25

This research was supported by National Science Foundation grant MCS76-11281, and the Office of Naval Research grant number N00014-67-A-0204-0065.

* Note: This paper was submitted prior to the time that Rivest became editor of the department, and editorial consideration was completed under the former editor, G. K. Manacher.

Authors' Address: MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139.

1. Introduction

The era of "electronic mail" [10] may soon be upon us; we must ensure that two important properties of the current "paper mail" system are preserved: (a) messages are *private*, and (b) messages can be *signed*. We demonstrate in this paper how to build these capabilities into an electronic mail system.

At the heart of our proposal is a new encryption method. This method provides an implementation of a "public-key cryptosystem", an elegant concept invented by Diffie and Hellman [1]. Their article motivated our research, since they presented the concept but not any practical implementation of such a system. Readers familiar with [1] may wish to skip directly to Section V for a description of our method.

II. Public-Key Cryptosystems

In a "public-key cryptosystem" each user places in a public file an encryption procedure E. That is, the public file is a directory giving the encryption procedure of each user. The user keeps secret the details of his corresponding decryption procedure D. These procedures have the following four properties:

(a) Deciphering the enciphered form of a message M yields M. Formally,

$$D(E(M)) = M. \quad (1)$$

(b) Both E and D are easy to compute.

(c) By publicly revealing E the user does not reveal an easy way to compute D. This means that in practice only he can decrypt messages encrypted with E, or compute D efficiently.

(d) If a message M is first deciphered and then enciphered, M is the result. Formally,

$$E(D(M)) = M. \quad (2)$$

An encryption (or decryption) procedure typically consists of a *general method* and an *encryption key*. The general method, under control of the key, enciphers a message M to obtain the enciphered form of the message, called the *ciphertext* C. Everyone can use the same general method; the security of a given procedure will rest on the security of the key. Revealing an encryption algorithm then means revealing the key.

When the user reveals E he reveals a very inefficient method of computing D(C): testing all possible messages M until one such that E(M) = C is found. If property (c) is satisfied the number of such messages to test will be so large that this approach is impractical.

A function E satisfying (a)–(c) is a "trap-door one-way function;" if it also satisfies (d) it is a "trap-door one-way permutation." Diffie and Hellman [1] introduced the concept of trap-door one-way functions but

Public Key

MATHEMATICAL GAMES

A new kind of cipher that would take millions of years to break

by Martin Gardner

"Few persons can be made to believe that it is not quite an easy thing to invent a method of secret writing which shall baffle investigation. Yet it may be roundly asserted that human ingenuity cannot concoct a cipher which human ingenuity cannot resolve."
—EDGAR ALLAN POE

The upward creep of postal rates accompanied by the deterioration of postal service is a trend that may or may not continue, but as far as most private communication is concerned, in a few decades it probably will not matter. The reason is simple. The transfer of information will probably be much faster and much cheaper by "electronic mail" than by conventional postal systems. Before long it should be possible to go to any telephone, insert a message into an attachment and dial a number. The telephone at the other end will print out the message at once.

Government agencies and large businesses will presumably be the first to make extensive use of electronic mail, followed by small businesses and private individuals. When this starts to happen, it will become increasingly desirable to have fast, efficient ciphers to safeguard information from electronic eavesdroppers. A similar problem is involved in protecting private information stored in computer memory banks from snoopers who have access to the memory through data-processing networks.

It is hardly surprising that in recent years a number of mathematicians have asked themselves: Is it possible to devise a cipher that can be rapidly encoded and decoded by computer, can be used repeatedly without changing the key and is unbreakable by sophisticated cryptanalysis? The surprising answer is yes. The breakthrough is scarcely two years old, yet it bids fair to revolutionize the entire field of secret communication. Indeed, it is so revolutionary that all previous ciphers, together with the techniques for cracking them, may soon fade into oblivion.

A Caesar cipher with a 10-shift

An unbreakable code can be unbreakable in theory or unbreakable only in practice. Edgar Allan Poe, who fancied himself a skilled cryptanalyst, was convinced that no cipher could be invented that could not also be "unriddled." Poe was certainly wrong. Ciphers that are unbreakable even in theory have been in use for half a century. They are "one-time pads," ciphers that are used only once, for a single message. Here is a simple example based on a shift cipher, sometimes called a Caesar cipher because Julius Caesar used it.

First write the alphabet, followed by the digits 0 through 9. (For coding purposes 0 represents a space between words, and the other digits are assigned to punctuation marks.) Below this write the same sequence cyclically shifted to the right by an arbitrary number of units, as is shown in color in the illustration on this page. Our cipher consists in taking each symbol in the plaintext (the message), finding it in the top row, and replacing it with the symbol directly below it. The result is a simple substitution cipher, easily broken by any amateur.

In spite of its simplicity, a shift cipher can be the basis of a truly unbreakable code. The trick is simply to use a different shift cipher for each symbol in the plaintext, each time choosing the amount of shift at random. This is easily done with the spinner shown in the top illustration on the opposite page. Suppose the first word of plaintext is THE. We spin the arrow and it stops on K. This tells us to use for encoding T a Caesar cipher in which the lower alphabet is shifted 10 steps to the right, bringing A below K as is shown in the illustration. T, therefore, is encoded as J. The same procedure is followed for every symbol in the plaintext. Before each symbol is encoded, the arrow is spun and the lower sequence is shifted accordingly. The result is a ciphertext starting with J and a cipher "key" starting with K. Note that the cipher key will be the same length as the plaintext.

To use this one-time cipher for sending a message to someone—call him Z—we must first send Z the key. This can be done by a trusted courier. Later we send to Z, perhaps by radio, the ciphertext. Z decodes it with the key and then destroys the key. The key must not be used again because if two such ciphertexts were intercepted, a cryptanalyst might have sufficient structure for breaking them.

It is easy to see why the one-time cipher is uncrackable even in principle. Since each symbol can be represented by any other symbol, and each choice of representation is completely random, there is no internal pattern. To put it another way, any message whatever having the same length as the ciphertext is as legitimate a decoding as any other. Even if the plaintext of such a coded message is found, it is of no future help to the cryptanalyst because the next time the system is used the randomly chosen key will be entirely different.

One-time pads are in constant use today for special messages between high military commanders, and between governments and their high-ranking agents. The "pad" is no more than a long list of random numbers, perhaps printed on many pages. The sender and receiver must of course have duplicate copies. The sender uses page 1 for a cipher, then destroys the page. The receiver uses his page 1 for decoding, then destroys his page. When the Russian agent Rudolf Abel was captured in New York in 1957, he had a one-time pad in the form of a booklet about the size of a postage stamp. David Kahn, who tells the story in his marvelous history *The Codebreakers*, says that the one-time pad is the standard method of secret radio communication used by the U.S.S.R. The famous "hot line" between Washington and Moscow also makes use of a one-time pad, the keys being periodically delivered through the two embassies.

If the one-time pad provides absolute secrecy, why is it not used for all secret communication? The answer is that it is too impractical. Each time it is employed a key must be sent in advance, and the key must be at least as long as the anticipated message. "The problem of producing, registering, distributing and canceling the keys," writes Kahn, "may seem slight to an individual who has not had experience with military communications, but in wartime the volumes of traffic stagger even the signal staffs. Hundreds of thousands of words may be enciphered in a day, simply to generate the millions of key characters required would be enormously expensive and time-consuming. Since each message must have its unique key, application of the ideal system would require shipping out on tape at the very least the equivalent of the total communications volume of a war."

Let us qualify Poe's dictum by applying it only to ciphers that are used repeatedly without any change in the key. Until recently all cipher systems of this kind were known to be theoretically breakable provided the code breaker has enough time and enough ciphertext. Then in 1975 a new kind of cipher was proposed that radically altered the situation by supplying a new definition of "unbreakable," a definition that comes from the branch of computer science known as complexity theory. These new ciphers are not absolutely unbreakable in the sense of the one-time pad, but in practice they are unbreakably in a much stronger sense than any cipher previously designed for widespread use. In principle these new ciphers can be broken, but only by computer programs that run for millions of years.

The two men responsible for this remarkable breakthrough are Whitfield Diffie and Martin E. Hellman, both electrical engineers at Stanford University. Their work was partly supported by the National Science Foundation in 1975 and was reported in their paper "New Directions in Cryptography" (*IEEE Transactions on Information Theory*, November, 1976). In it Diffie and Hellman show how to create unbreakable ciphers that do not require advance sending of a key or even concealment of the method of encoding. The ciphers can be efficiently encoded and decoded, they can be used over and over again and there is a bonus: the system also provides an "electronic signature" that, unlike a written signature, cannot be forged. If Z receives a "signed" message from A, the signature proves to Z that A actually sent the message. Moreover, A's signature cannot be forged by an eavesdropper or even by Z himself!


Randomizer for making a "one-time pad"

These seemingly impossible feats are made possible by what Diffie and Hellman call a trapdoor one-way function. Such a function has the following properties: (1) it will change any positive integer x to a unique positive integer y; (2) it has an inverse function that changes y back to x; (3) efficient algorithms exist for computing both the forward function and its inverse; (4) if only the function and its forward algorithm are known, it is computationally infeasible to discover the inverse algorithm.

The last property is the curious one that gives the function its name. It is like a trapdoor: easy to drop through but hard to get up through. Indeed, it is impossible to get up through the door unless one knows where the secret button is hidden. The button symbolizes the "trapdoor information." Without it one cannot open the door from below, but the button is so carefully concealed that the probability of finding it is practically zero.

Before giving a specific example, let us see how such functions make the new cryptographic systems possible. Suppose there is a group of businessmen who want to communicate secrets to one another. Each devises his own trapdoor function with its forward and backward algorithms. A handbook is published in which each company's encoding (forward) algorithm is given in full. The decoding (inverse) algorithms are kept secret. The handbook is public. Anyone can consult it and use it for sending a secret message to any listed company.

Suppose you are not a member of the group but you want to send a secret message to member Z. First you change your plaintext to a long number, using a standard procedure given in the handbook. Next you look up Z's forward algorithm and your computer uses it for rapid encoding of the ciphertext. This new number is sent to Z. It does not matter at all if the ciphertext is overheard or intercepted because only Z knows his secret decoding procedure. There is no way a curious cryptanalyst, studying Z's public encoding algorithm, can discover Z's decoding algorithm. In principle he might find it, but in practice that would require a supercomputer and a few million years of running time.

An outsider cannot "sign" a message to Z, but any member of the group can. Here is the devilishly clever way the signature works. Suppose A wants to sign a message to Z. He first encodes the plaintext number by using his own secret inverse algorithm. Then he encodes the ciphertext number a second time, using Z's public algorithm. After Z receives the ciphertext he first transforms it by applying his own secret decoding algorithm, then he applies A's public encoding algorithm. Out comes the message!

Z knows that only A could have sent this doubly encoded ciphertext because it made use of A's secret algorithm. A's "signature" is clearly unforgeable. Z cannot use it to send a message purporting to come from A because Z still does

9686	9613	7546	2206
1477	1409	2225	4355
6829	0575	9991	1246
7431	9874	6951	2093
0816	2982	2514	5708
3969	3147	6622	8839
8962	8013	3919	9055
1829	9461	5781	5154

A ciphertext challenge worth $100

L. Rivest*

ing
l Public-

Adleman
cience

with the novel
yption key
ing decryption
ces:
not needed to
nciphered
ed by the
r the message,
decryption key,
privately held
signature using
cryption key,
or cannot later
has obvious
lectronic funds
ned by
M to a publicly
remainder
ly specified
mbers p and q,
secret, power d
(q − 1)). The
the difficulty of

natures, public-
ion, security,
mail, message-
iography,
81, 5-25

ssing or research of
final readers and to
© ACM's copyright
e publication, to its
ssages were granted
ing Machinery. To
ital concept, or the
n republication, or

Science Foundation
search grant number

he time that Rivest
consideration was
ather.
ipurse Science, 545

1. Introduction

The era of "electronic mail" [1,9] may soon be upon us; we must ensure that two important properties of the current "paper mail" system are preserved: (a) messages are *private*, and (b) messages can be *signed*. We demonstrate in this paper how to build these capabilities into an electronic mail system.

At the heart of our proposal is a new encryption method. This method provides an implementation of a "public-key cryptosystem", an elegant concept invented by Diffie and Hellman [3]. Their article motivated our research, since they presented the concept but not any practical implementation of such a system. Readers familiar with [3] may wish to skip directly to Section V for a description of our method.

II. Public-Key Cryptosystems

In a "public-key cryptosystem" each user places in a public file an encryption procedure E. That is, the public file is a directory giving the encryption procedure of each user. The user keeps secret the details of his corresponding decryption procedure D. These procedures have the following four properties:

(a) Deciphering the enciphered form of a message M yields M. Formally,

$$D(E(M)) = M. \quad (1)$$

(b) Both E and D are easy to compute.

(c) By publicly revealing E the user does not reveal an easy way to compute D. This means that in practice only he can decrypt messages encrypted with E, or compute D efficiently.

(d) If a message M is first deciphered and then enciphered, M is the result. Formally,

$$E(D(M)) = M. \quad (2)$$

An encryption (or decryption) procedure typically consists of a *general method* and an *encryption key*. The general method, under control of the key, enciphers a message M to obtain the enciphered form of the message, called the *ciphertext* C. Everyone can use the same general method; the security of a given procedure will rest on the security of the key. Revealing an encryption algorithm then means revealing the key.

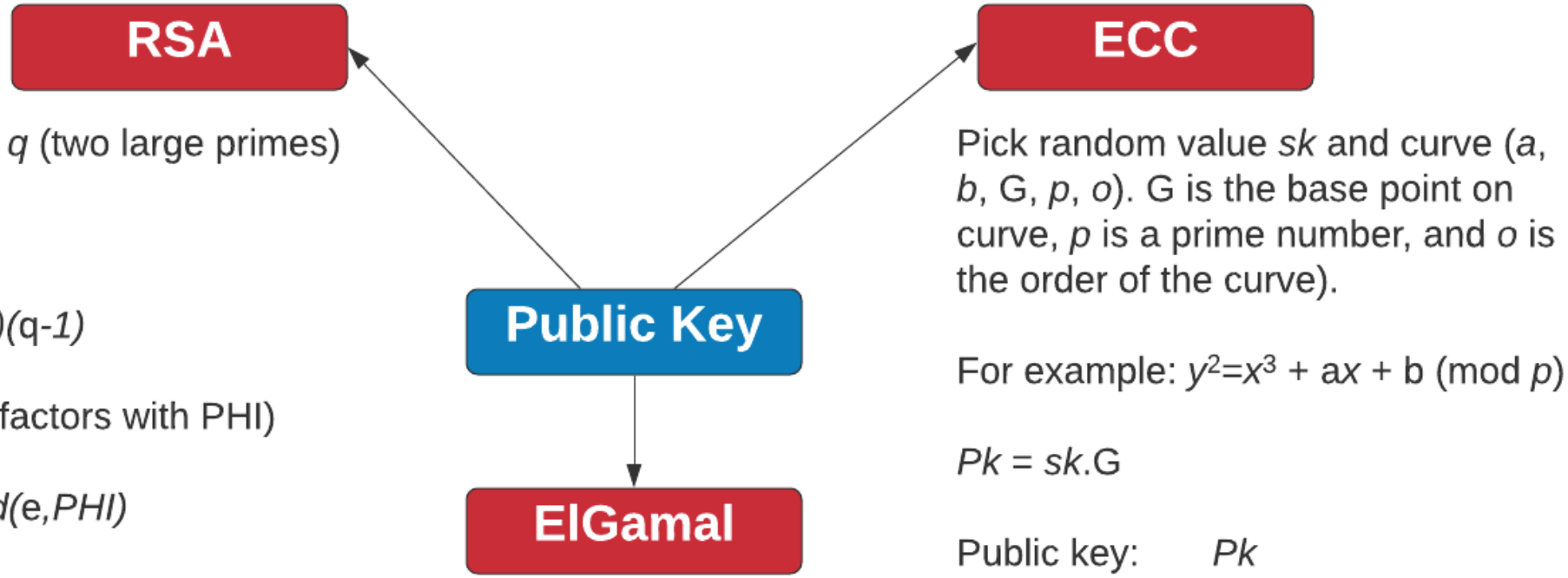
When the user reveals E he reveals a very inefficient method of computing D(C): testing all possible messages M until one such that E(M) = C is found. If property (c) is satisfied the number of such messages to test will be so large that this approach is impractical.

A function E satisfying (a)–(c) is a "trap-door one-way function;" if it also satisfies (d) it is a "trap-door one-way permutation." Diffie and Hellman [1] introduced the concept of trap-door one-way functions but

Communications of the ACM

February 1978
Volume 21
Number 2

Public Key



Pick p and q (two large primes)

$N = p.q$

$PHI = (p-1)(q-1)$

Pick e (no factors with PHI)

$d = InvMod(e,PHI)$

Public: (e,N)
Private: (d,N)

$C = M^e \pmod N$
$P = C^d \pmod N$

Pick random value x, generator (g) and prime (p)

$Y = g^x \pmod p$

Public key: (Y,p)
Private: (x,p)

Pick random value sk and curve (a, b, G, p, o). G is the base point on curve, p is a prime number, and o is the order of the curve).

For example: $y^2=x^3 + ax + b \pmod p$

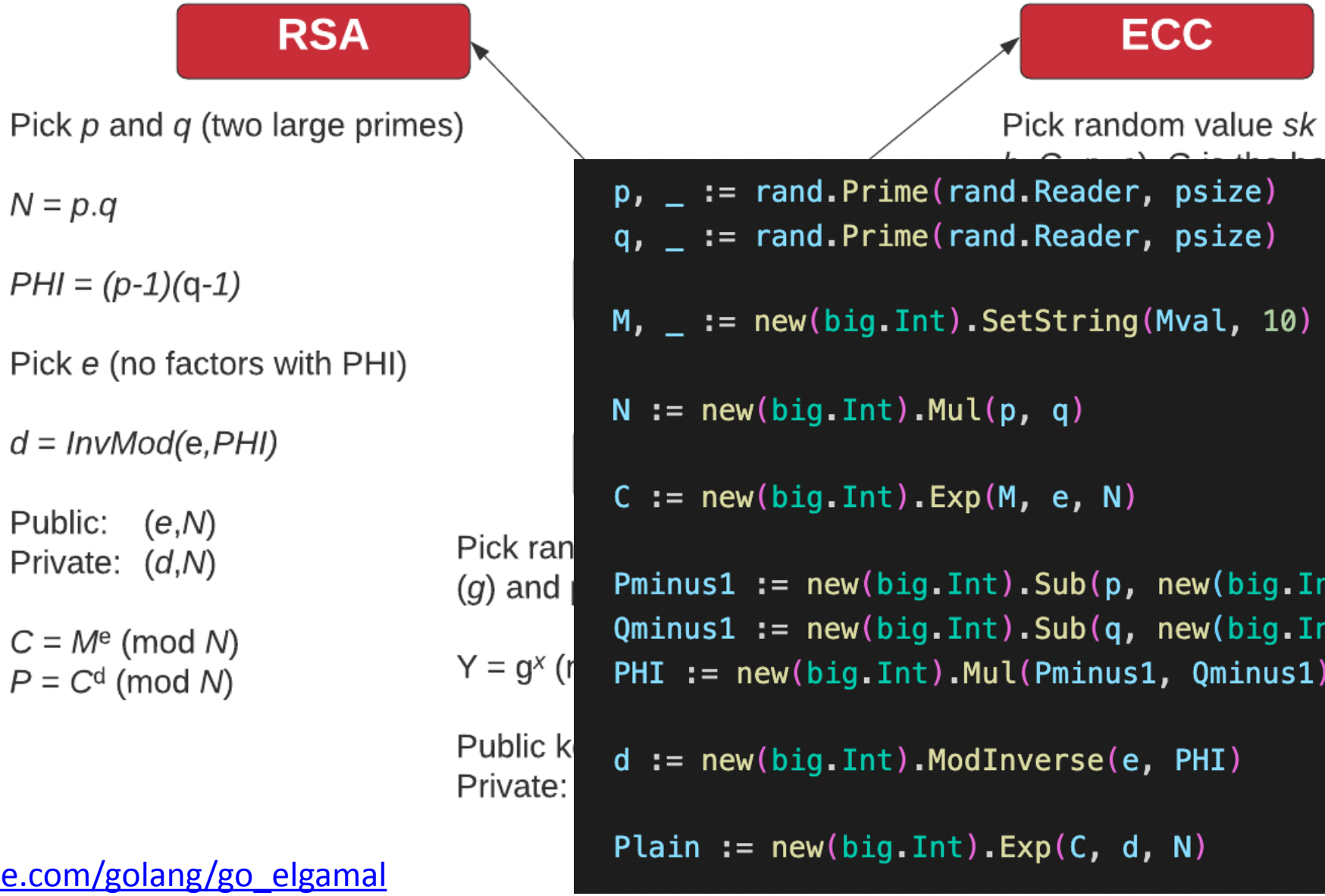
$Pk = sk.G$

Public key: Pk
Private key: sk

https://asecuritysite.com/golang/go_elgamal

ECC was co-invented by Neal Koblitz and Victor S Miller
ElGamal was invented by Tahir ElGamal

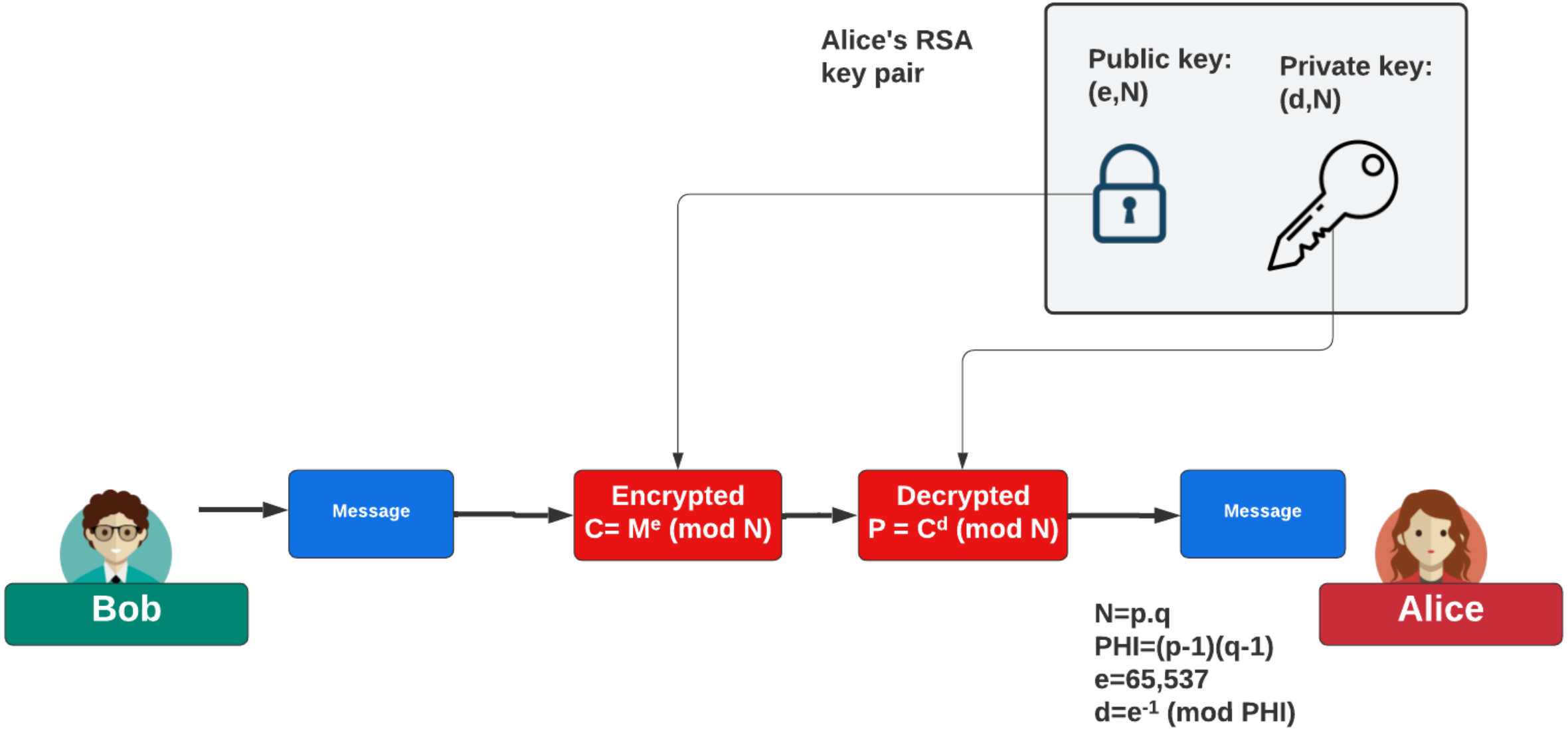
RSA




```go
p, _ := rand.Prime(rand.Reader, psize)
q, _ := rand.Prime(rand.Reader, psize)

M, _ := new(big.Int).SetString(Mval, 10) // Base 10

N := new(big.Int).Mul(p, q)

C := new(big.Int).Exp(M, e, N)

Pminus1 := new(big.Int).Sub(p, new(big.Int).SetInt64(1))
Qminus1 := new(big.Int).Sub(q, new(big.Int).SetInt64(1))
PHI := new(big.Int).Mul(Pminus1, Qminus1)

d := new(big.Int).ModInverse(e, PHI)

Plain := new(big.Int).Exp(C, d, N)
```

https://asecuritysite.com/golang/go_elgamal

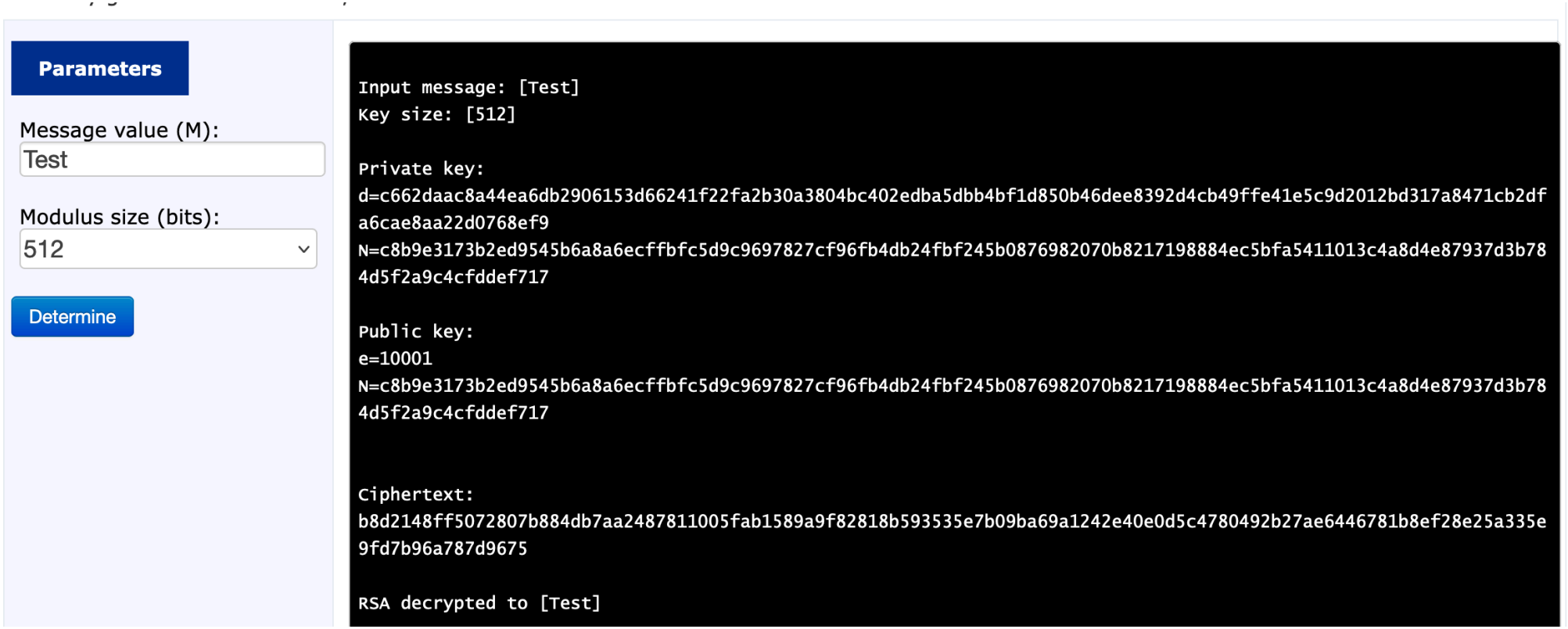
RSA


Alice's RSA
key pair
Public key:
(e,N)
Private key:
(d,N)
Bob
Message
Encrypted
C= M^e (mod N)
Decrypted
P = C^d (mod N)
Message
Alice
N=p.q
PHI=(p-1)(q-1)
e=65,537
d=e^-1 (mod PHI)

Optimal Asymmetric Encryption Padding (OAEP) padding

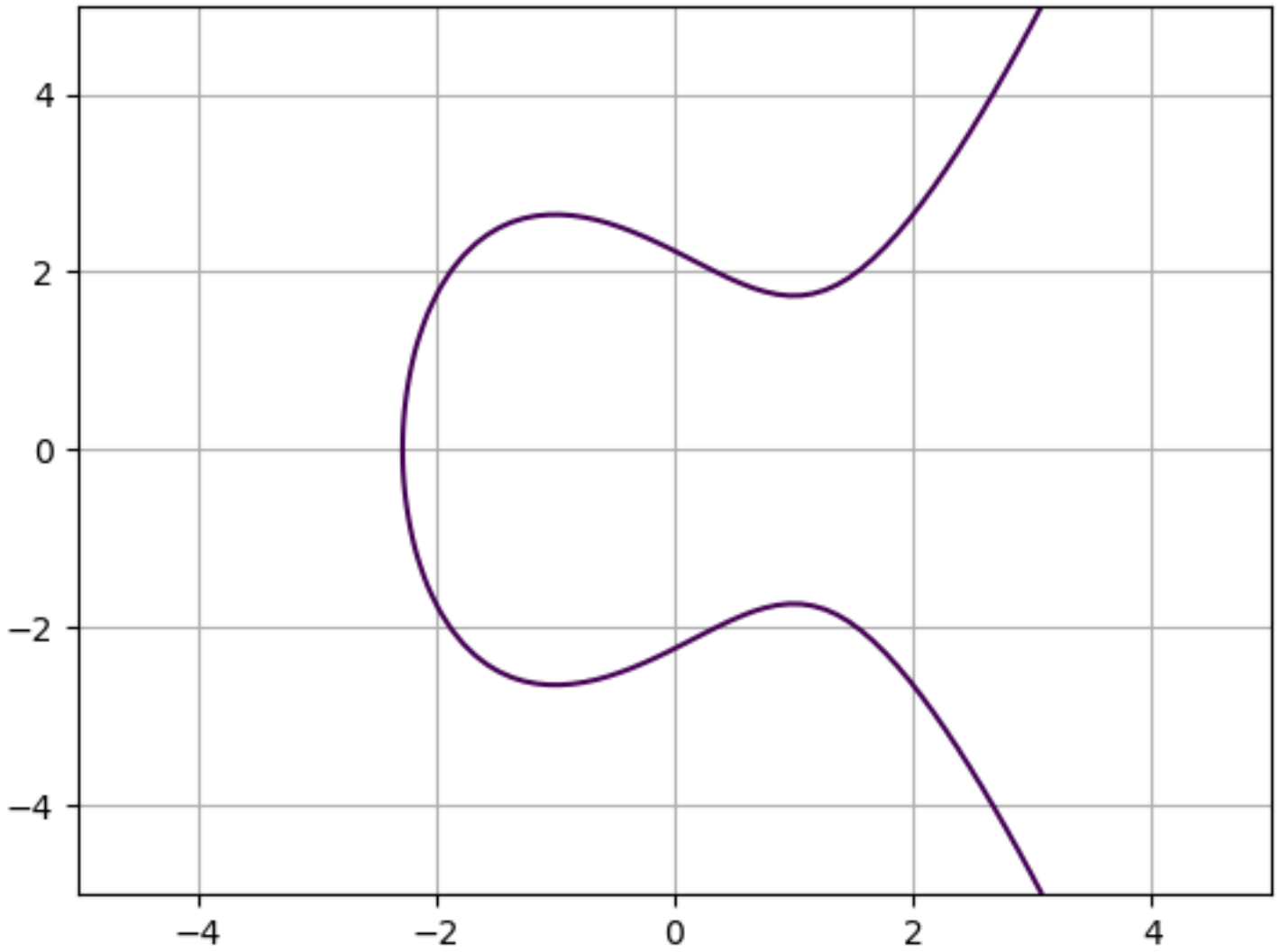
https://asecuritysite.com/rsa/go_rsa2

RSA: PKCS1v15 padding



https://asecuritysite.com/rsa/go_rsa3

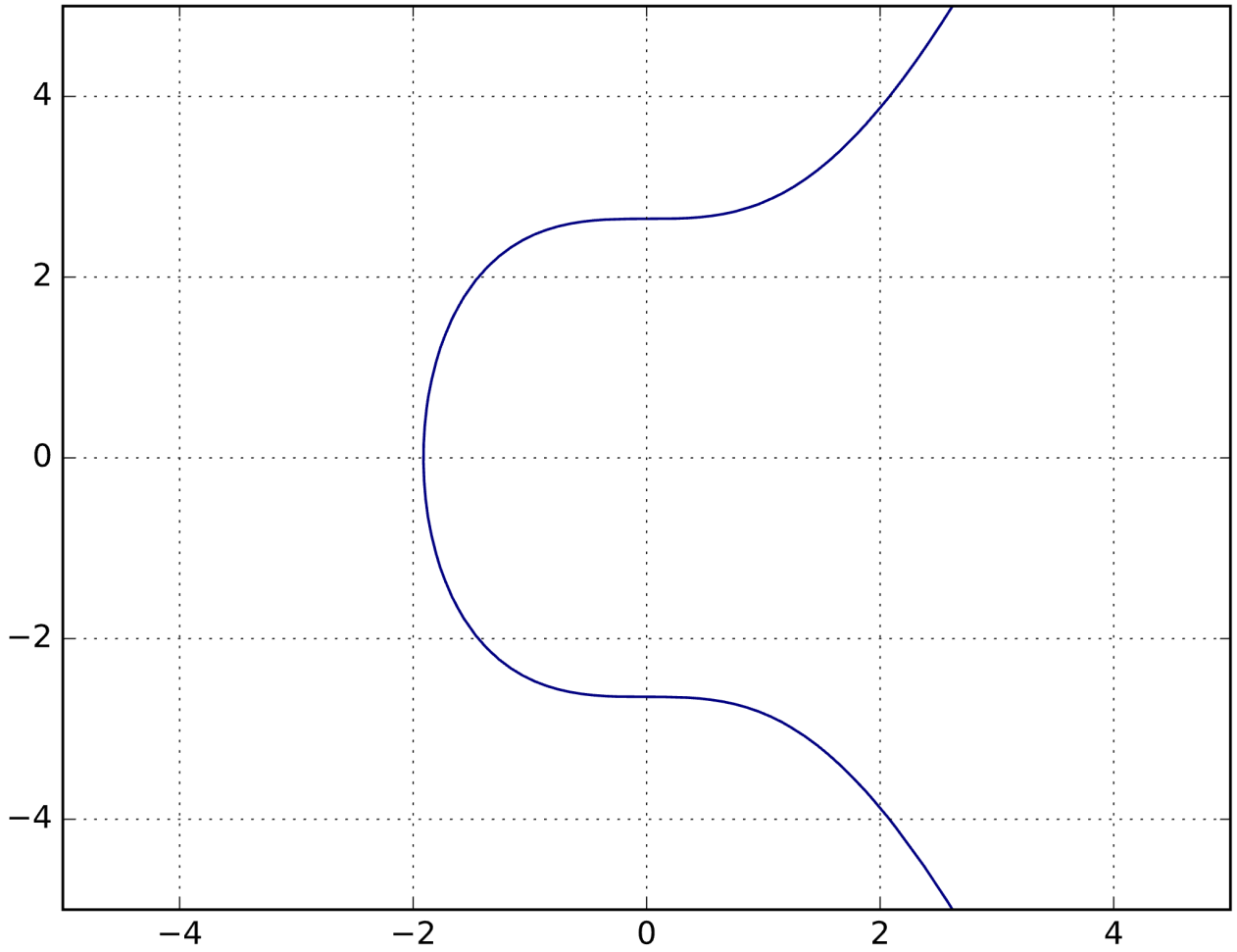
Crypto – Elliptic Curve



$Y^2=x^3-3x+5$

https://asecuritysite.com/ecc/plot05

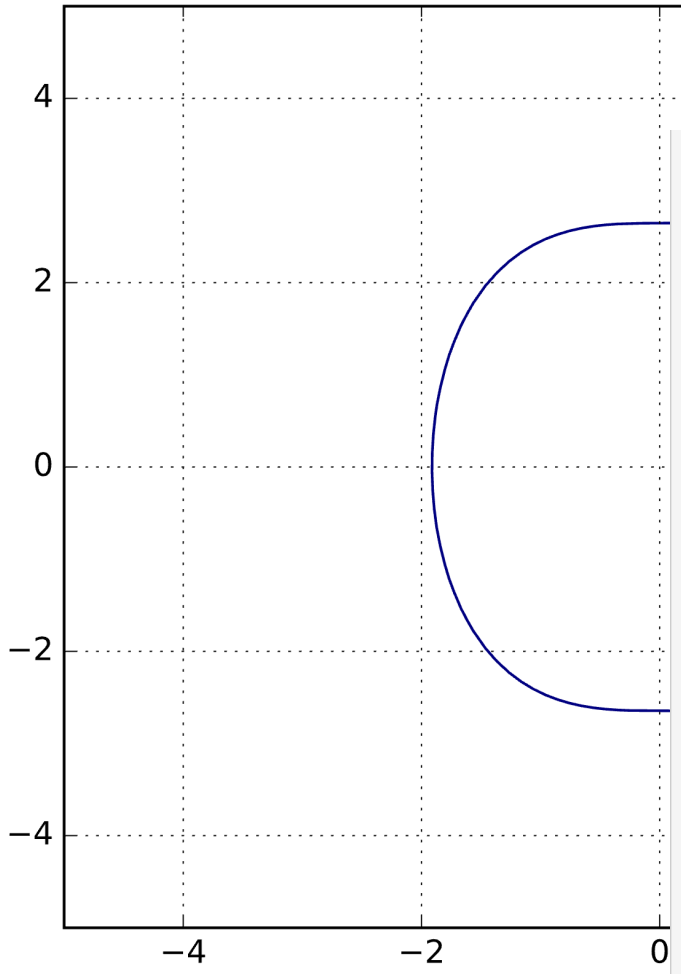
Crypto – Elliptic Curve



Private key:
0xc9f4f55bdeb5ba0bd337f2dbc952a5439e20ef9a
f6203d25d014e7102d86aaeeL
Public key:
0xc44370819cb3b7b57b2aa7edf550a9a5410c234
d27aff497458bbbfec8b6a327,
0x52a1a3e222cd89cbd2764b69bd9b0ea5c4fd6ca
28861e1f2140eeff9c2e76487

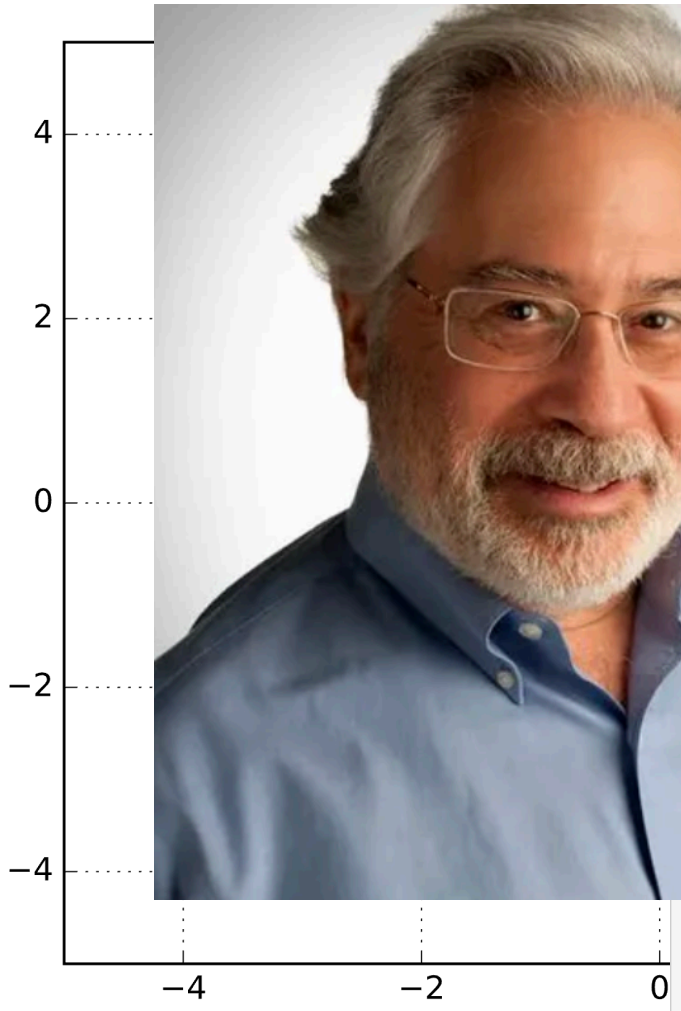
G:
(506626302227734366957871889516853432625
060345377759417550018736038911672924 0L,
326705100207588169780830851305070431844 7
127338065924327593890433575733748242 4L)

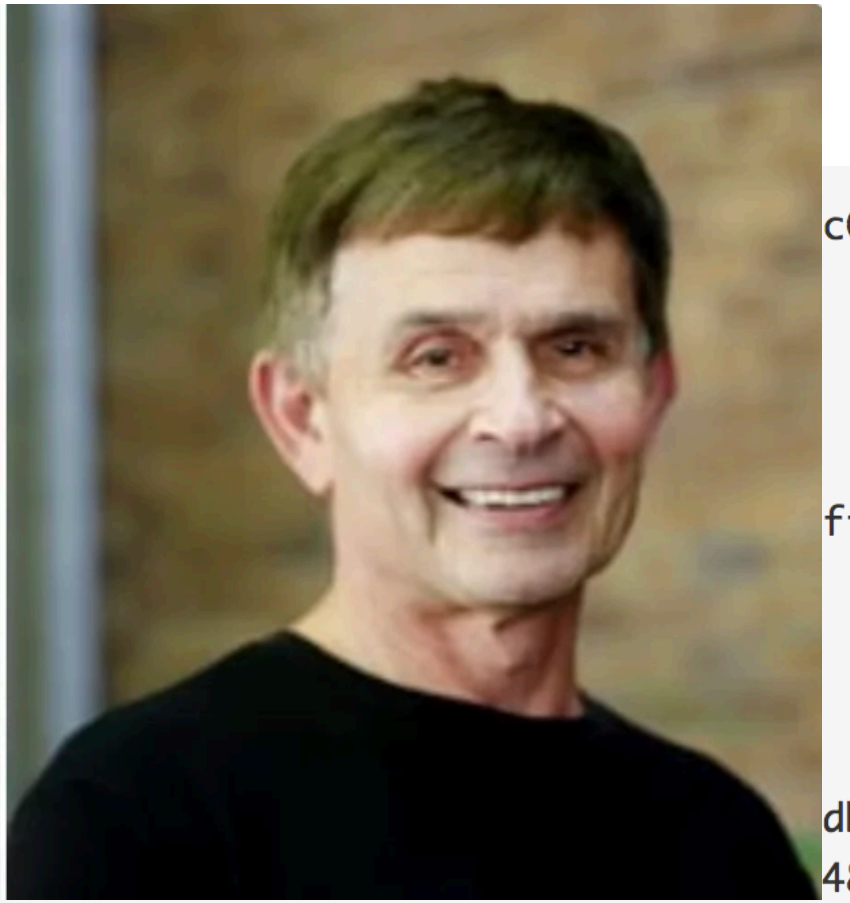
Crypto – Elliptic Curve




```
EllipticCurve = collections.namedtuple('EllipticCurve', 'name p a b g n h')

curve = EllipticCurve(
    'secp256k1',
    # Field characteristic.
    p=0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f,
    # Curve coefficients.
    a=0,
    b=7,
    # Base point.
    g=(0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f81798,
       0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10d4b8),
    # Subgroup order.
    n=0xfffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141,
    # Subgroup cofactor.
    h=1,
)
```

Crypto – Elliptic Curve





```
cCurve', 'name p a b g n h')

ffffffffffffffffffffffffefffffc2f,

db2dce28d959f2815b16f81798,
48a68554199c47d08ffb10d4b8),
    # Subgroup order.
    n=0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd0364141,
    # Subgroup cofactor.
    h=1,
)
```

Crypto – Elliptic Curve

MATHEMATICS OF COMPUTATION
VOLUME 48, NUMBER 177
JANUARY 1987, PAGES 203–209

Elliptic Curve Cryptosystems

By Neal Koblitz

This paper is dedicated to Daniel Shanks on the occasion of his seventieth birthday

Abstract. We discuss analogs based on elliptic curves over finite fields of public key cryptosystems which use the multiplicative group of a finite field. These elliptic curve cryptosystems may be more secure, because the analog of the discrete logarithm problem on elliptic curves is likely to be harder than the classical discrete logarithm problem, especially over GF(2^n). We discuss the question of primitive points on an elliptic curve modulo p, and give a theorem on nonsmoothness of the order of the cyclic subgroup generated by a global point.

```
rve', 'name p a b g n h')

ffffffffffffefffffc2f,

dce28d959f2815b16f81798,
68554199c47d08ffb10d4b8),

48a03bbfd25e8cd0364141,

    # Subgroup cofactor.
    h=1,
)
```

Crypto – Elliptic Curve

MATHEMATICS OF COMPUTATION
VOLUME 48, NUMBER 177
JANUARY 1987, PAGES 203–209

Use of Elliptic Curves in Cryptography

Victor S. Miller

Exploratory Computer Science, IBM Research, P.O. Box 218, Yorktown Heights, NY 10598

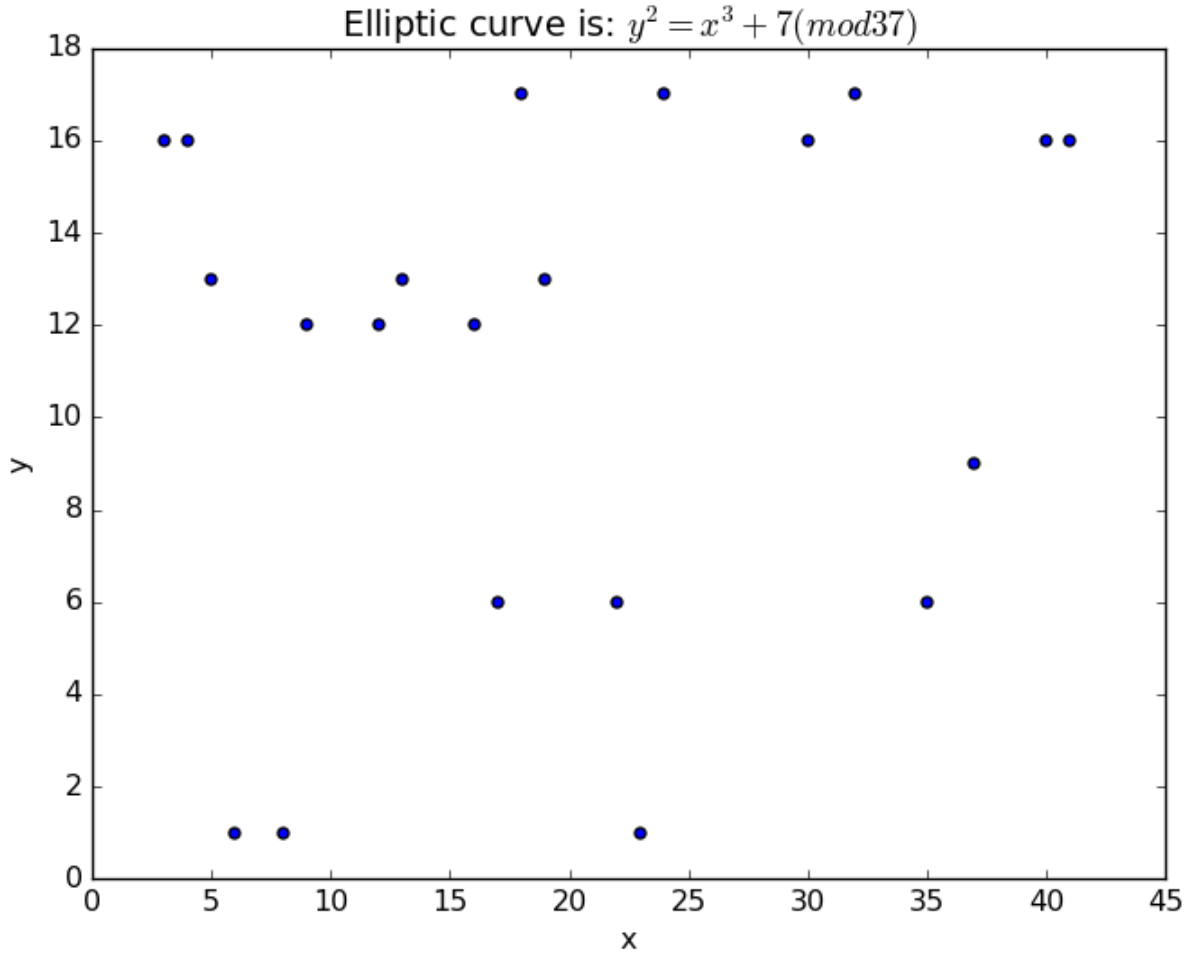
ABSTRACT

We discuss the use of elliptic curves in cryptography. In particular, we propose an analogue of the Diffie-Hellmann key exchange protocol which appears to be immune from attacks of the style of Western, Miller, and Adleman. With the current bounds for infeasible attack, it appears to be about 20% faster than the Diffie-Hellmann scheme over GF(p). As computational power grows, this disparity should get rapidly bigger.

Elliptic Curve

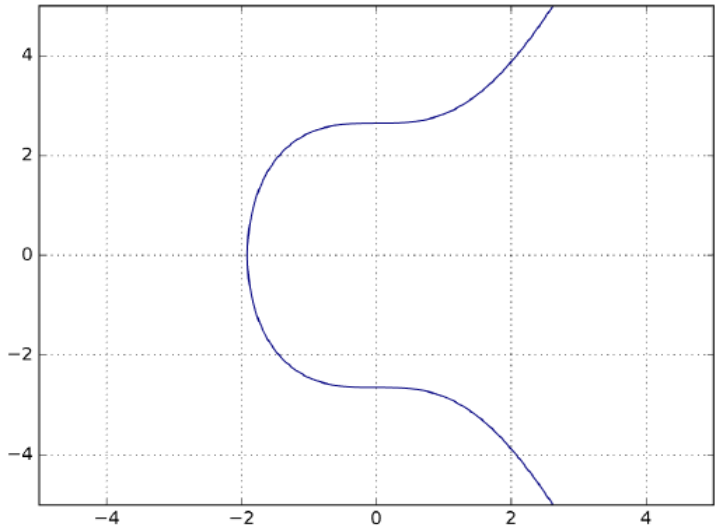
- A: 0
- B: 7
- Prime number: 37
- Elliptic curve is: $y^2=x^3+7$
- Finding the first 20 points

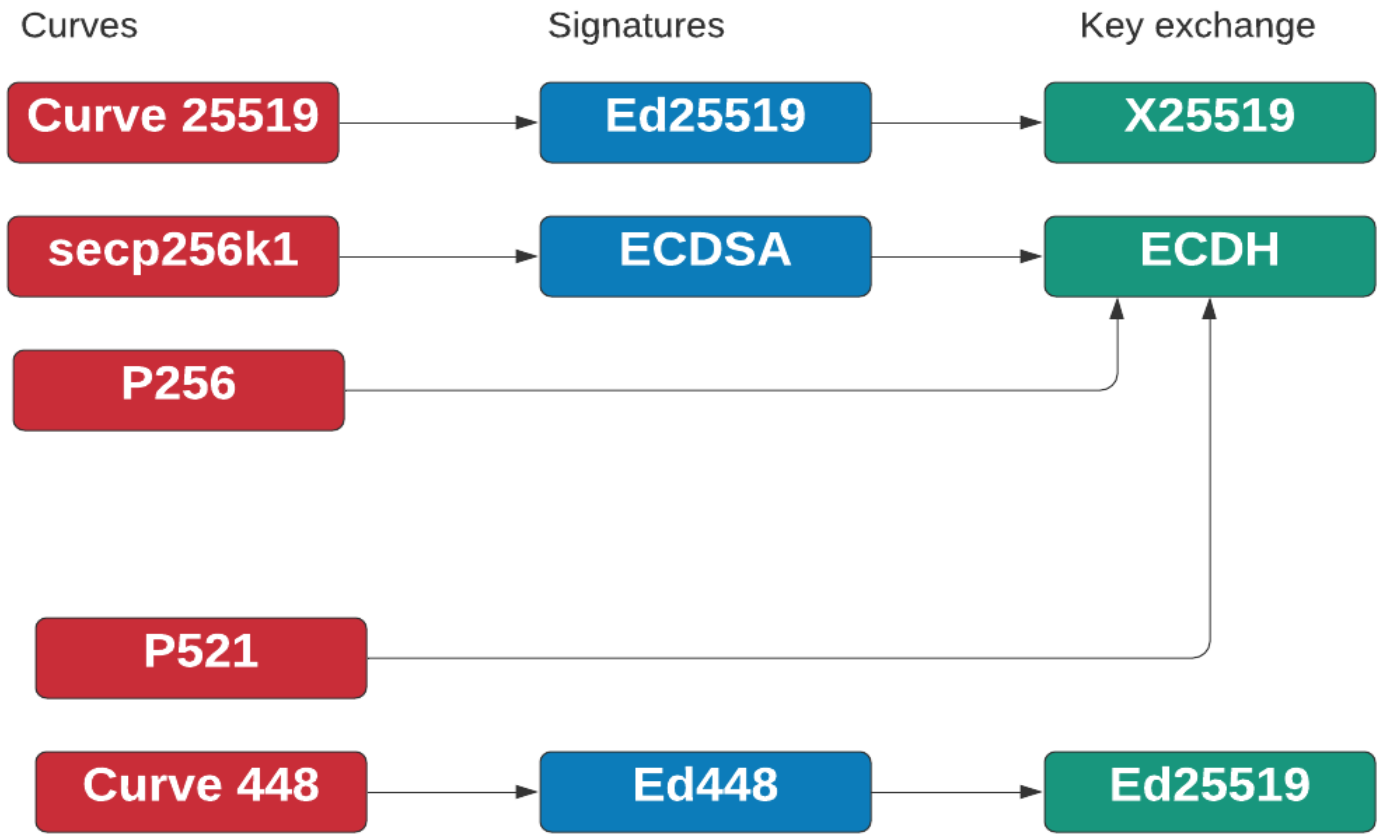
- (3 , 16) (4 , 16) (5 , 13) (6 , 1) (8 , 1
  (9 , 12) (12 , 12) (13 , 13) (16 , 12)
  (17 , 6) (18 , 17) (19 , 13) (22 , 6)
  (23 , 1) (24 , 17) (30 , 16) (32 , 17)
  (35 , 6) (37 , 9) (40 , 16) (41 , 16)
  [Link]



https://asecuritysite.com/ecc/ecc_pointsv

Elliptic Curve Types

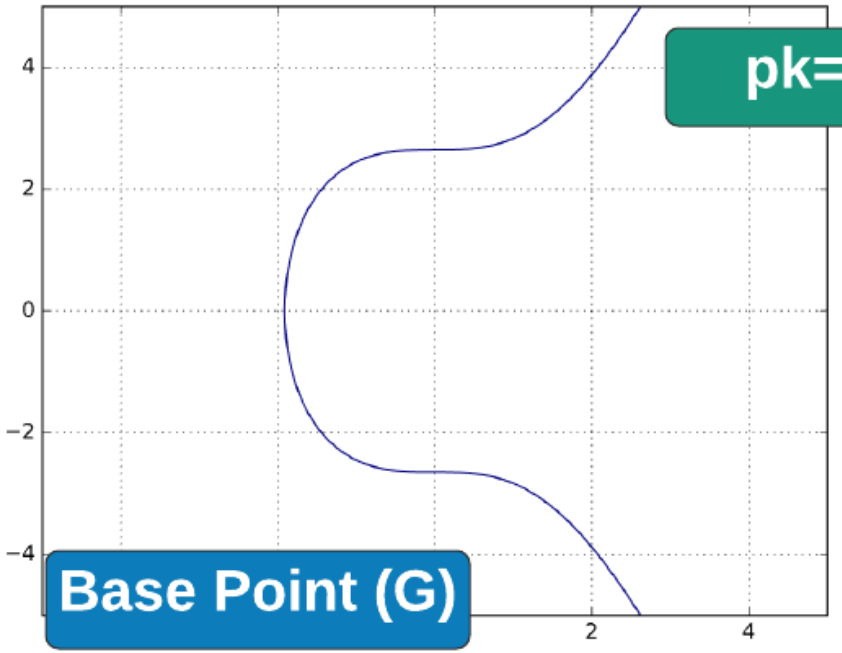




https://asecuritysite.com/ecc/ecc_types

Crypto – Elliptic Curve Types

$y^2=x^3+ax+b \pmod p$



Base Point (G)

pk=sk.G

Bob

sk (private key)

Basic operations:
- Point add.
- Point double.

https://asecuritysite.com/ecc/ecc_types

ECC Keys

```
C \> openssl ec -in priv.pem -text -noout
read EC key
Private-Key  (256 bit)
priv
    46 b9 e8 61 b6 3d 35 09 c8 8b 78 17 27 5a 30
    d2 2d 62 c8 cd 8f a6 48 6d de e3 5e f0 d8 e0
    49 5f
pub
    04 25 00 e7 f3 fb dd f2 84 29 03 f5 44 dd c8
    74 94 ce 95 02 9a ce 4e 25 7d 54 ba 77 f2 bc
    1f 3a 88 37 a9 46 1c 4f 1c 57 fe cc 49 97 53
    38 1e 77 2a 12 8a 58 20 a9 24 a2 fa 05 16 2e
    b6 62 98 7a 9f
ASN1 OID  secp256k1
```

https://asecuritysite.com/ecc/ecc_types

ECC Parameters

```
C:> openssl ecparam -in priv.pem -text -param_enc explicit -noout
Field Type: prime-field
Prime:
    00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:fe:ff:
    ff:fc:2f
A:    0
B:    7 (0x7)
Generator (uncompressed):
    04:79:be:66:7e:f9:dc:bb:ac:55:a0:62:95:ce:87:
    0b:07:02:9b:fc:db:2d:ce:28:d9:59:f2:81:5b:16:
    f8:17:98:48:3a:da:77:26:a3:c4:65:5d:a4:fb:fc:
    0e:11:08:a8:fd:17:b4:48:a6:85:54:19:9c:47:d0:
    8f:fb:10:d4:b8
Order:
    00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:
    ff:fe:ba:ae:dc:e6:af:48:a0:3b:bf:d2:5e:8c:d0:
    36:41:41
Cofactor:  1 (0x1)
```

https://asecuritysite.com/ecc/ecc_types

Crypto – Compressed Point

We can reduce our public key point by compressing it. With a compressed point, we get a 32 byte compressed value (64 hex characters), and then add a 02 or a 03 to represent when y is even (02) or odd (03). This can be seen when we compress our public key here:

```
Private key: f91d8f3a49805fff9289769247e984b355939679f3080156fe295229e00f25af
Public key: 52972572d465d016d4c501887b8df303eee3ed602c056b1eb09260dfa0da0ab288742f4dc97d9edb6fd946babc002fdfb06f26caf117b9405ed79275763fdb1c
    Compressed public key 0252972572d465d016d4c501887b8df303eee3ed602c056b1eb09260dfa0da0ab2
```

In this case, we can see that we have a 02 at the start of the compressed public key point, and followed by the x-coordinate. This means that our y-coordinate value is even. Now let's look at another one:

```
Private key: ac609e0cc9681f8cb63e968be20e0f19721751561944f5b4e52d54d5f27ec57b
Public key: 18ed2e1ec629e2d3dae7be1103d4f911c24e0c80e70038f5eb5548245c475f504c220d01e1ca419cb1ba4b3393b615e99dd20aa6bf071078f70fd949008e7411
    Compressed public key 0318ed2e1ec629e2d3dae7be1103d4f911c24e0c80e70038f5eb5548245c475f50
```

We can see that we now have a 03 at the start, followed by the x-coordinate value from the point. But how do we get the 02 or 03? Well in the first case the y-axis co-ordinate ends with a hex value of 'c' (1100), and which is even. In this case, we append a 02. In the second case, the y-coordinate ends with a hex value of '1' (0001), and is then odd. In this case, we add a 03. The compressed key is then:

```
02 (x-co-ordinate) // if y is even
03 (x-co-ordinate) // if y is odd
```

When we need to uncompress, we can easily compute the y-coordinate value again from the x-coordinate value, and examine whether y is odd or even. This will recover the uncompressed point. Typically an uncompressed point will start with a 04.

In Ethereum, we also identify a user with their Ethereum address. This is a 160-bit (40 hex characters) address, and is derived from a Keccak 256 bits hash of the uncompressed x and y coordinate values. A sample run of this is:

```
Private key: f91d8f3a49805fff9289769247e984b355939679f3080156fe295229e00f25af
Public key: 52972572d465d016d4c501887b8df303eee3ed602c056b1eb09260dfa0da0ab288742f4dc97d9edb6fd946babc002fdfb06f26caf117b9405ed79275763fdb1c
    Compressed public key 0252972572d465d016d4c501887b8df303eee3ed602c056b1eb09260dfa0da0ab2
    Signer address 0x6eDBe1F6D48FbF1b053D6c9FA7997C710B84f55F
```

https://asecuritysite.com/ecc/js_ethereum2

Key Formats

```
Private key encoding:    Encoding.DER
Private key format:      PrivateFormat.PKCS8
Public key encoding:     Encoding.DER
Public key format:       PublicFormat.PKCS1
Curve:                   RSA
Key size:                512

Private key:
308201540201003000d06092a864886f70d0101010500048201 3e3082013a020100024100bfcf2c5b831cd6579189bab8418cb096372d101c
668d99cd5079c7b58de99c3dd5ea2fab4a4bf19a41f1143aaf82452ea8609544986f0117984afcdbdac154650203010001024046850 5daf3
5e83d398379a05aa5bee849bee9e70c2a7a6f75c7219a1bf763731c5ceb1230260ff50b091c81c9b270b47def7003b89c266995fefbf018a
1a3e55022100e523d1939db160871ff34adce4e3962c0748daf75b2333f3abf99ffba04dc407022100d64b1cae0f642ebc95ff18c7fe0683
6b1becd83f2c2b7cf2b9b8de4d4736c13302200869863a2b21e5ef5006f88a9f849370ce5ba85e48644475ffac4c694a7b5065022100bfe7
b01d68872d1b91b04b3efe8c52b05ee7b1989133b8c79deea295ce0a88e5022074a097eb0daff57e361d72f38c0a2b73fd68df0aab1e6984
447812fee25cd7e6

Public key:
3048024100bfcf2c5b831cd6579189bab8418cb096372d101c668d99cd5079c7b58de99c3dd5ea2fab4a4bf19a41f1143aaf82452ea86095
44986f0117984afcdbdac154650203010001

== Key details ==
Private key p: 10364292841754088417087161716913261840592498736222044253218359323126159990887 1
Private key q: 96927661068226713248491942945113782574452825748964486405373918841092196188467
Private key d:
369341221961886833851721498469919956183256415184304476108168409578339377934529232324700580129991672783168019389 6
536781822416345057489540698633556754841173
Private key dmp1: 380494813269761153722503487967000442322131802788504790778390413528869813053 3
Private key dmq1: 868011112515712620436393507705376794680048879748344554546499732484505087858 93
Private key iqmp: 527520344667507198077077896837049374089710678831746212374600302562278887157 50
Public key N:
100458666377738856319320489085798481711484024411428771324760460919425722397575121700349731799150222932926240977 1
442014001670944207908040991988922074119075 7
Public key e: 65537
```

https://asecuritysite.com/openssh/openssh

Key Formats

```
Private key encoding:    Encoding.DER
Private key format:      PrivateFormat.PKCS8
Public key encoding:     Encoding.DER
Public key format:       PublicFormat.PKCS1
Curve:                   RSA
Key size:                512

Private key:
308201540201003000d06092a864886f70d0101010500048201e3082013a0201000241
668d99cd5079c7b58de99c3dd5ea2fab4a4bf19a41f1143aaf82452ea8609544986f01
5e83d398379a05aa5bee849bee9e70c2a7a6f75c7219a1bf763731c5ceb1230260ff50
1a3e55022100e523d1939db160871ff34adce4e3962c0748daf75b2333f3abf99ffba0
6b1becd83f2c2b7cf2b9b8de4d4736c13302200869863a2b21e5ef5006f88a9f849370
b01d68872d1b91b04b3efe8c52b05ee7b1989133b8c79deea295ce0a88e5022074a097
447812fee25cd7e6

Public key:
3048024100bfcf2c5b831cd6579189bab8418cb096372d101c668d99cd5079c7b58de9
44986f0117984afcdbdac154650203010001

== Key details ==
Private key p: 10364292841754088417087161716913261840592498736222044425
Private key q: 96927661068226713248491942945113782574452825748964486400
Private key d:
369341221961886833851721498469919956183256415184304476108168409578339353678182241634505748954069863355675484117 3
Private key dmp1: 380494813269761153722503487967000442322131802788504 7
Private key dmq1: 868011112515712620436393507705376794680048879748344 5
Private key iqmp: 527520344667507198077077896837049374089710678831746212374600302562278887157500
Public key N:
100458666377738856319320489085798481711484024411428771324760460919425722397575121700349731799150222932926240977144201400167094420790804099198892207411907 57
Public key e: 65537
```

```
Private key encoding:    Encoding.PEM
Private key format:      PrivateFormat.PKCS8
Public key encoding:     Encoding.DER
Public key format:       PublicFormat.PKCS1
Curve:                   RSA
Key size:                512

Private key:
-----BEGIN PRIVATE KEY-----
MIIBVwIBADANBgkqhkiG9w0BAQEFAASCAUEwggE9AgEAAkEAw8fv2b5jgpA3iPqV
a/VQaSuL8MJO+ydrNEb8WoeHbyGnRue7AazElDxMNmSg2zBYKI5vwBNoBgfT5f7+
Zyh9nwIDAQABAkEAoHh/stTn0XuN2Xuy/8MZ/qoP0O4jOJ7tsvLAIx/VNZNI7C+w
I61nIYpjauInllsV9PsHzW7s6KQR5hLgYFXDCQIhAP7A/UeqgBVaRxYA63Qrg9+3
2eb0Z1bx488u77O7OrTDAiEAxLOZk8FUC6JIHJFqY5lhmX18pxCx/5cvB4frGArE
lfUCIQCA8gZBhBeYtz1L8eM7xdrWHiT5NbQo2f7JG6TQgL1CKQIhAIvHmvuKuU8G
vttlScR3/osSSFVenEjvPl/IZvUO+XtdAiEArTkmgEr1hGtgOojsO9lSyyBMhNNC
dZlYrS9I71mrZDY=
-----END PRIVATE KEY-----


Public key:
3048024100c3c7efd9be6382903788fa956bf550692b8bf0c24efb276b3446fc5a87876f21a746e7bb01acc4943c4c3664a0db3058288e6fc013680607d3e5fefe67287d9f0203010001

== Key details ==
```

https://asecuritysite.com/openssh/openssh

Key Formats: Distinguished Encoding Rules (DER) encoding of ASN.1

Details

DER:

```
3059301306072a8648ce3d020106
6082a8648ce3d03010703420004
2927b10512bae3eddcfe4678281
28bad2903269919f7086069c8c4
df6c732838c7787964eaac00e59
21fb1498a60f4606766b3d96850
```

Determine

The following are some examples:

- **Ex 1 (ECDSA P-256)** Try.
- **Ex 2 (ECDSA secp256k1)** Try.
- **Ex 3 (ECDSA secp256r1)** Try.
- **Ex 4 (ECDSA secp224r1)** Try.
- **Ex 5 (ECDSA Brainpool P224)** Try.
- **Ex 6 (Brainpool P224r1)** Try.
- **Ex 7 (secp521r1)** Try.

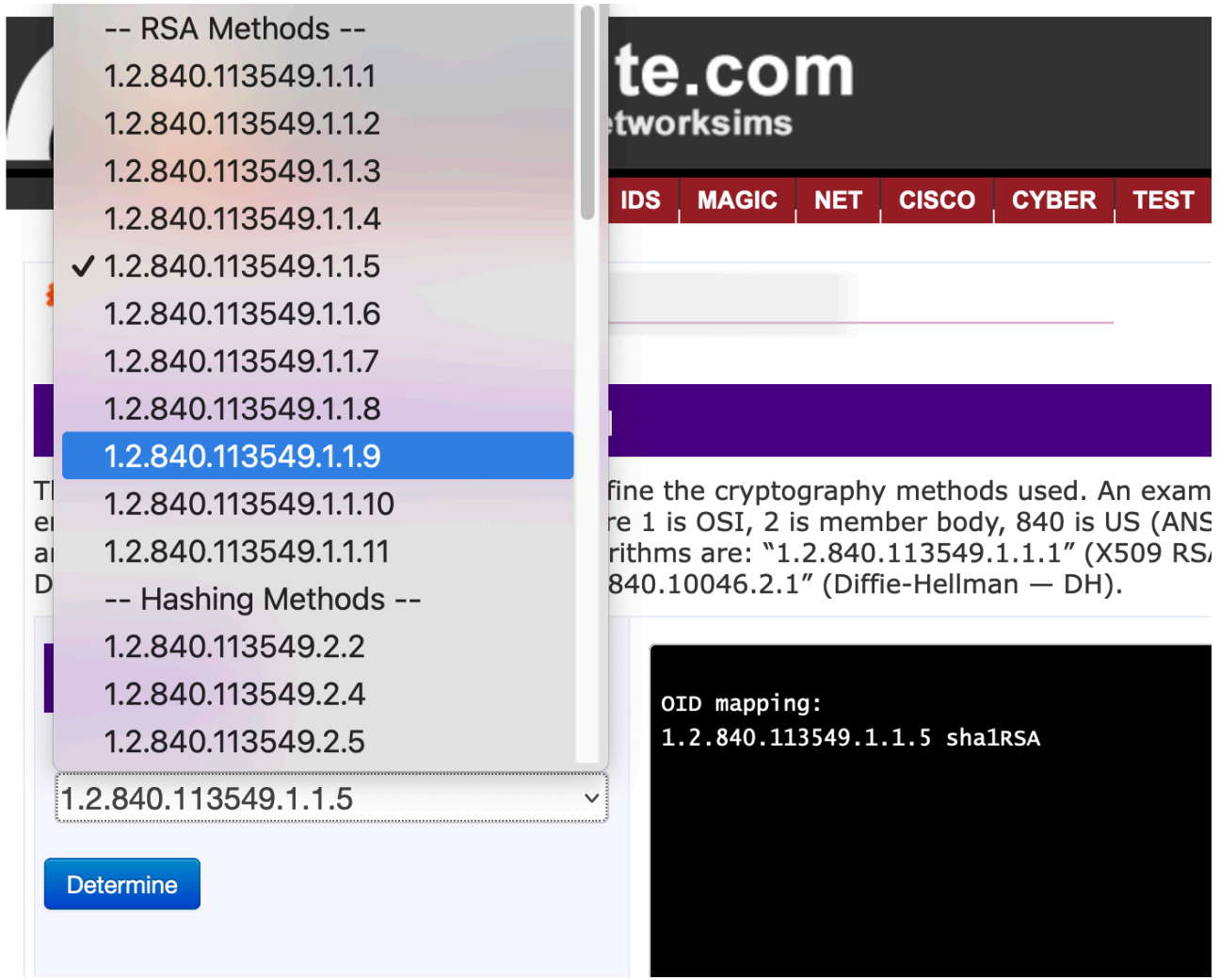
```
DER string:
3059301306072a8648ce3d020106082a8648ce3d03010703420004292 7b10512bae3eddcfe467828128bad2903269919f7086069c8
c4df6c732838c7787964eaac00e5921fb1498a60f4606766b3d9685001558d1a974e7341513e


==Sequence==
--->Sequence (30)
   --->Obj ID tag (06 - Object ID)
         ID algorithm: 1.2.840.10045.2.1 ECC (ecPublicKey)
   --->Obj ID tag (06 - Object ID)
         ID algorithm: 1.2.840.10045.3.1.7 secp256r1
--->Obj ID tag (03)
    Bit value:
b'042927b10512bae3eddcfe467828128bad2903269919f7086069c8c4df6c732838c7787964eaac00e5921fb1498a60f4606766b3
d9685001558d1a974e7341513e'
   Public key:
(186149555733158976576809766506854500809319199132692239587324523535938241925 68,902231163478598801665701987
253875695674142545475699253279885398331505739902 06)

Now checking key if ECC - 256

 EccKey(curve='NIST P-256',
point_x=186149555733158976576809766506854500809319199132692239587324523535938241925 68,
point_y=902231163478598801665701987253875695674142545475699253279885398331505739902 06)
```

(Abstract Syntax Notation One).

https://asecuritysite.com/digitalcert/sigs3

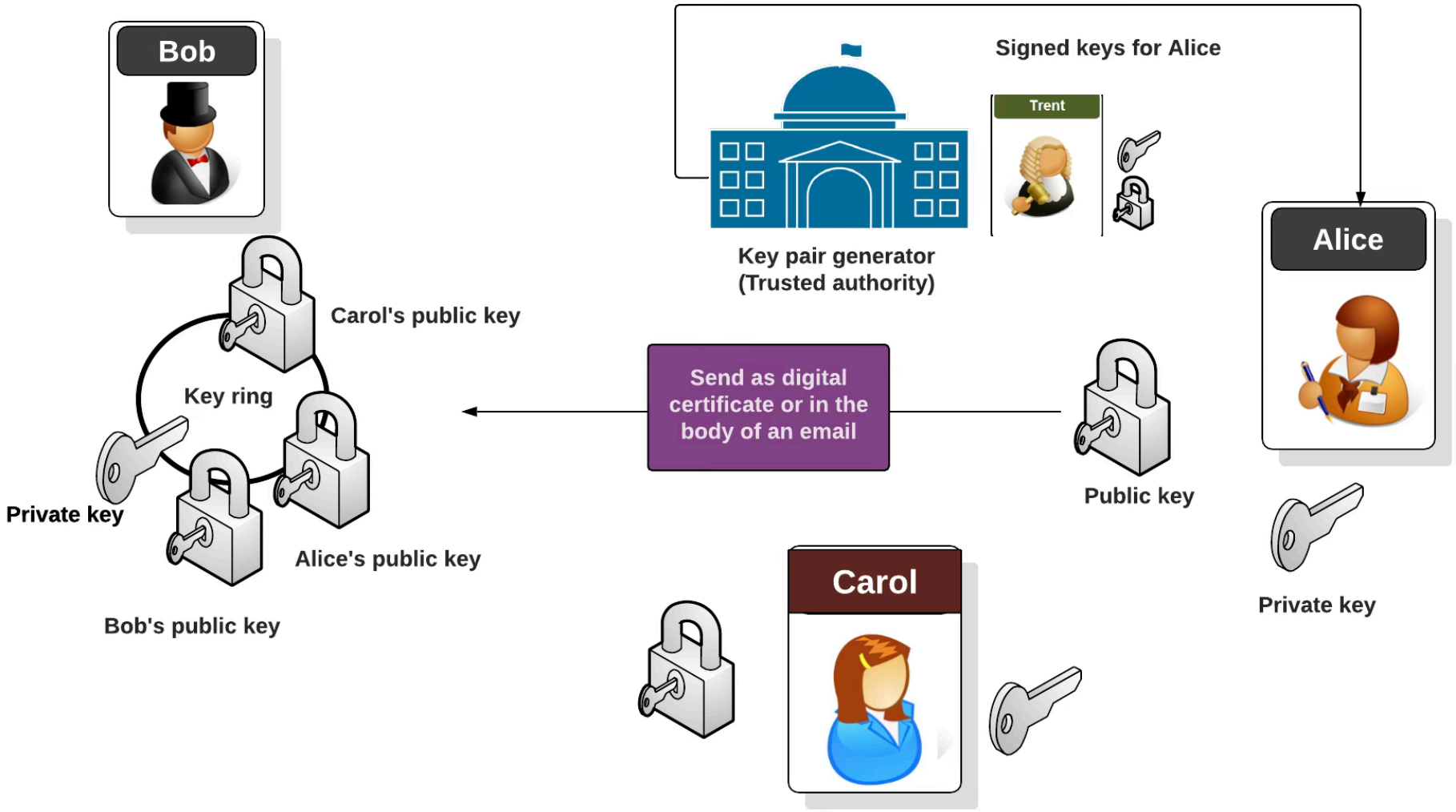
Object identifier (OID)

```
Hashing:
MD2      1.2.840.113549.2.2
MD5      1.2.840.113549.2.5
SHA-1    1.3.14.3.2.26
SHA-224 2.16.840.1.101.3.4.2.4
SHA-256 2.16.840.1.101.3.4.2.1
SHA-394 2.16.840.1.101.3.4.2.2
SHA-512 2.16.840.1.101.3.4.2.3Public key:
RSA Encryption  1.2.840.113549.1.1.1
DSA      1.2.840.10040.4.1
Diffie-Hellman (dhPublicNumber) 1.2.840.10046.2.1
ECC (ecPublicKey)       1.2.840.10045.2.1
md2WithRsaEncryption    1.2.840.113549.1.1.2Signatures:
md5WithRsaEncryption    1.2.840.113549.1.1.4
sha1WithRsaEncryption   1.2.840.113549.1.1.5
sha224WithRsaEncryption         1.2.840.113549.1.1.14
sha256WithRsaEncryption 1.2.840.113549.1.1.11
sha384WithRsaEncryption 1.2.840.113549.1.1.12
sha512WithRsaEncryption 1.2.840.113549.1.1.13
dsaWithSha1      1.2.840.10040.4.3
dsaWithSha224    2.16.840.1.101.3.4.3.1
dsaWithSha256    2.16.840.1.101.3.4.3.2
ecdsaWithSha1    1.2.840.10045.4.1
ecdsaWithSha224 1.2.840.10045.4.3.1
ecdsaWithSha256 1.2.840.10045.4.3.2
ecdsaWithSha384 1.2.840.10045.4.3.3
ecdsaWithSha512 1.2.840.10045.4.3.4Password Base Encryption
Algorithms:
pbeWithMd2AndDesCbc      1.2.840.113549.1.5.1
pbeWithMd5AndDesCbc      1.2.840.113549.1.5.3
pbeWithSha1AndDesCbc     1.2.840.113549.1.5.10
pbeWithMd2AndRc2Cbc      1.2.840.113549.1.5.4
pbeWithMd5AndRc2Cbc      1.2.840.113549.1.5.6
pbeWithSha1AndRc2Cbc     1.2.840.113549.1.5.11
pbeWithSha1And40BitRc2Cbc        1.2.840.113549.1.12.1.6
pbeWithSha1And128BitRc2Cbc       1.2.840.113549.1.12.1.5
pbeWithSha1And40BitRc4   1.2.840.113549.1.12.1.2
pbeWithSha1And128BitRc4 1.2.840.113549.1.12.1.1
pbeWithSha1And3DesCbc    1.2.840.113549.1.12.1.3Symmetric Encryption
Algorithms:
DES CBC 1.3.14.3.2.7
3DES CBC         1.2.840.113549.3.7
```



https://asecuritysite.com/powershell/oid

Key Ring


Bob
Signed keys for Alice
Trent
Key pair generator
(Trusted authority)
Alice
Carol's public key
Key ring
Send as digital certificate or in the body of an email
Public key
Private key
Alice's public key
Bob's public key
Carol
Private key

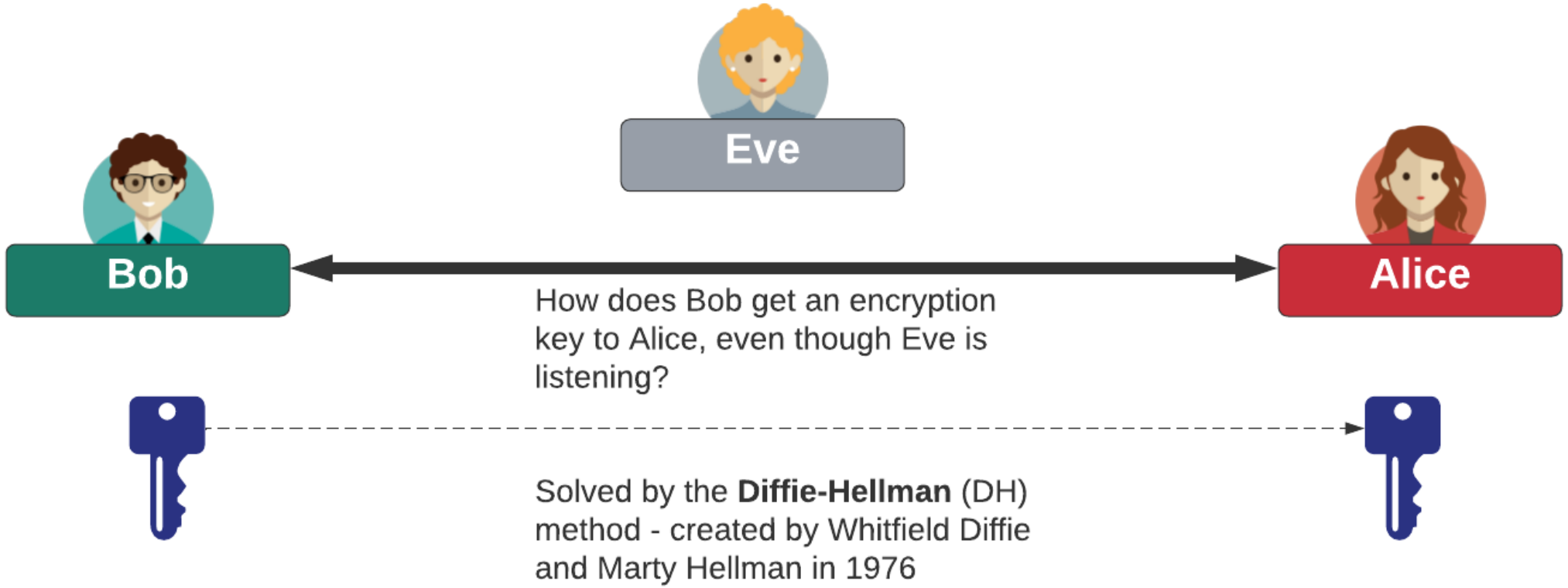
Cryptography: Key Exchange

Prof Bill Buchanan OBE FRSE

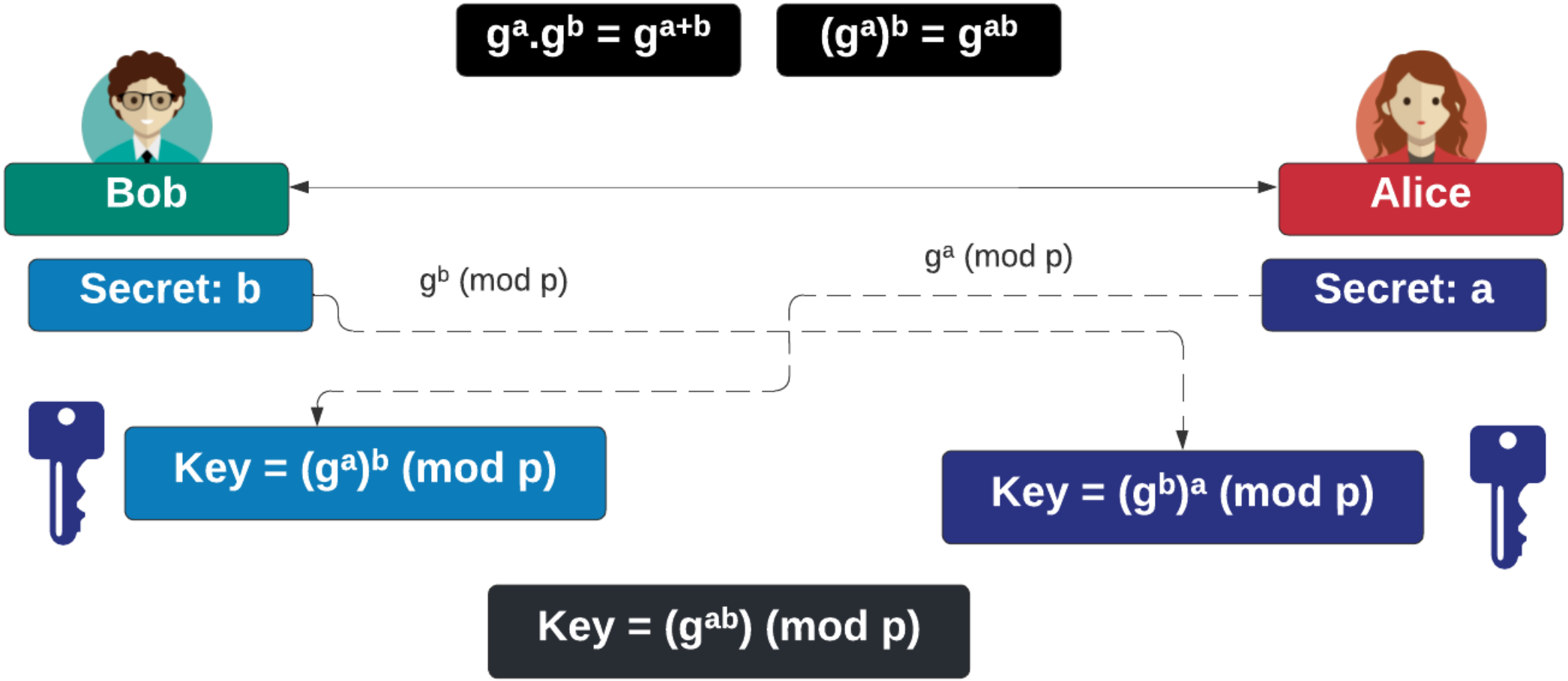
https://asecuritysite.com/keyexchange

Fundamentals
Symmetric Key
Hashing
MAC
KDF
Public Key
Key Exchange
Signatures
Digital Certificates

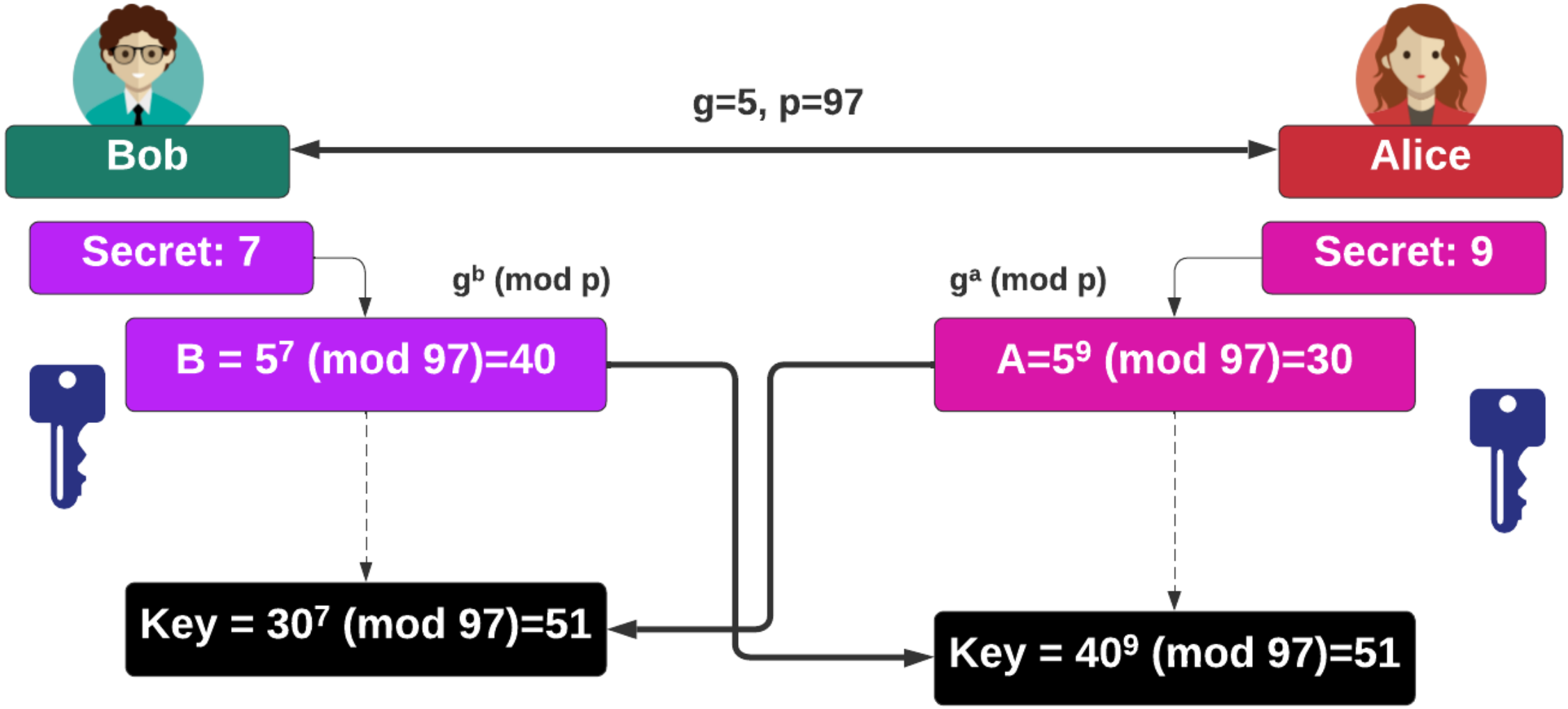
Key Exchange Problem



The Diffie-Hellman Method



The Diffie-Hellman Method



Picking g and p

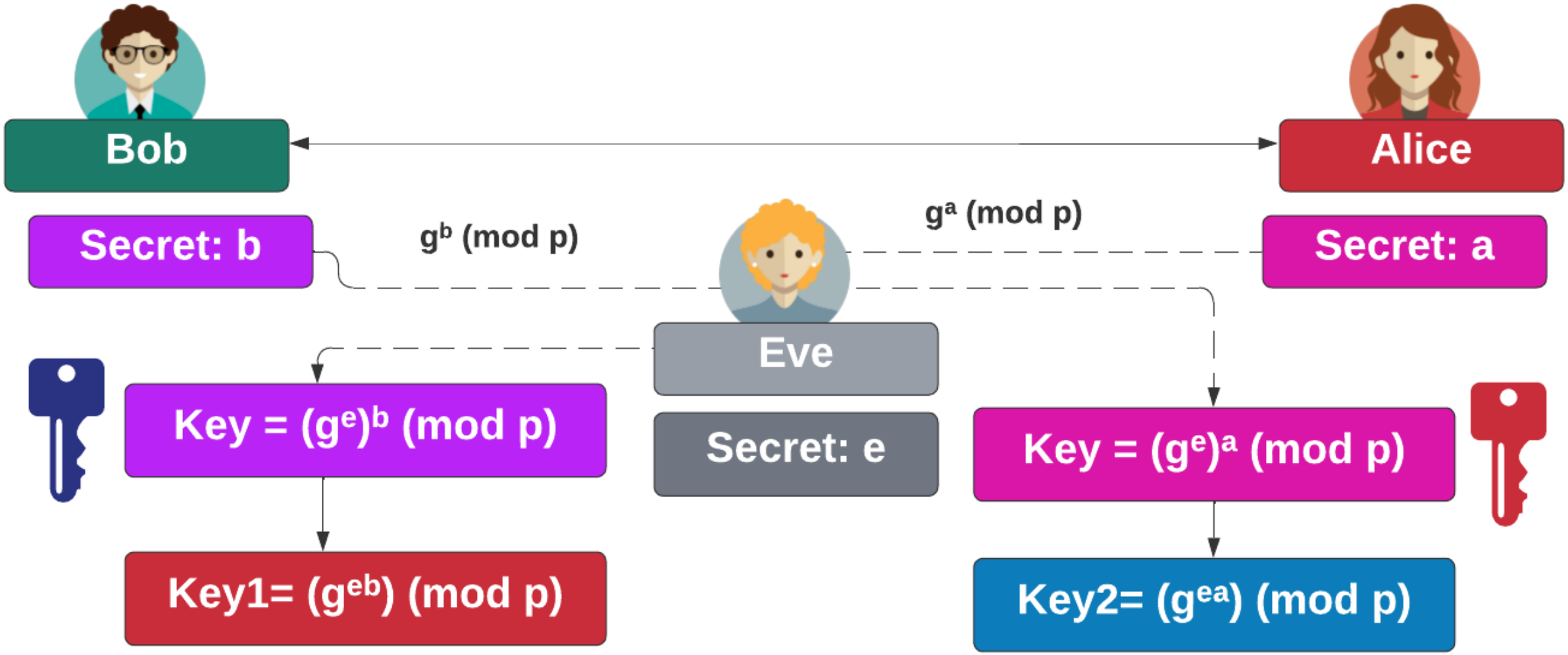
$Y = g^x \pmod p$

Let's try p=11

g	2	3	4	5	6	7	8	9
x	$g^x \pmod p$							
1	2	3	4	5	6	7	8	9
2	4	9	5	3	3	5	9	4
3	8	5	9	4	7	2	6	3
4	5	4	3	9	9	3	4	5
5	10	1	1	1	10	10	10	1
6	9	3	4	5	5	4	3	9
7	7	9	5	3	8	6	2	4
8	3	5	9	4	4	9	5	3
9	6	4	3	9	2	8	7	5
10	1	1	1	1	1	1	1	1

Picking G

Eve-in-the-Middle

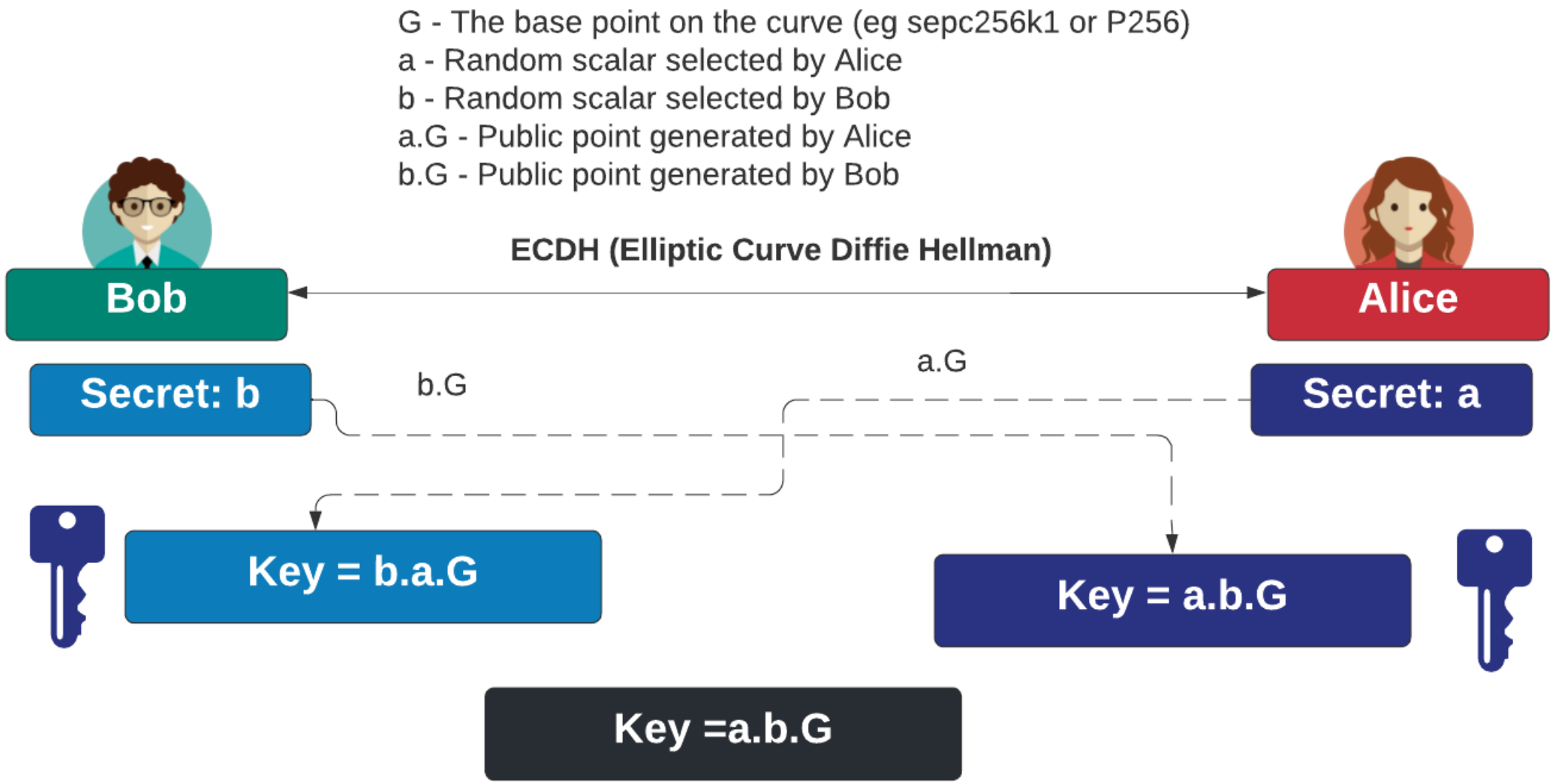


Diffie-Hellman Generation

```
C:\> openssl dhparam -out dhparams.pem 768 –text
C:\> type dhparams.pem
Diffie-Hellman-Parameters: (768 bit)
    prime:
        00:d0:37:c2:95:64:02:ea:12:2b:51:50:a2:84:6c:
        71:6a:3e:2c:a9:80:e2:65:b2:a5:ee:77:26:22:31:
        66:9e:fc:c8:09:94:e8:9d:f4:cd:bf:d2:37:b2:fb:
        b8:38:2c:87:28:38:dc:95:24:73:06:d3:d9:1f:af:
        78:01:10:6a:7e:56:4e:7b:ee:b4:8d:6b:4d:b5:9b:
        93:c6:f1:74:60:01:0d:96:7e:85:ca:b8:1f:f7:bc:
        43:b7:40:4d:4e:87:e3
    generator: 2 (0x2)
-----BEGIN DH PARAMETERS-----
MGYCYQDQN8KVZALqEitRUKKEbHFqPiypgOJlsqXudyYiMWae/
MgJlOid9M2/0jey
+7g4LIcoONyVJHMG09kfr3gBEGp+Vk577rSNa021m5PG8XRgAQ2WfoXKuB/
3vEO3
QE1Oh+MCAQI=
-----END DH PARAMETERS-----
```

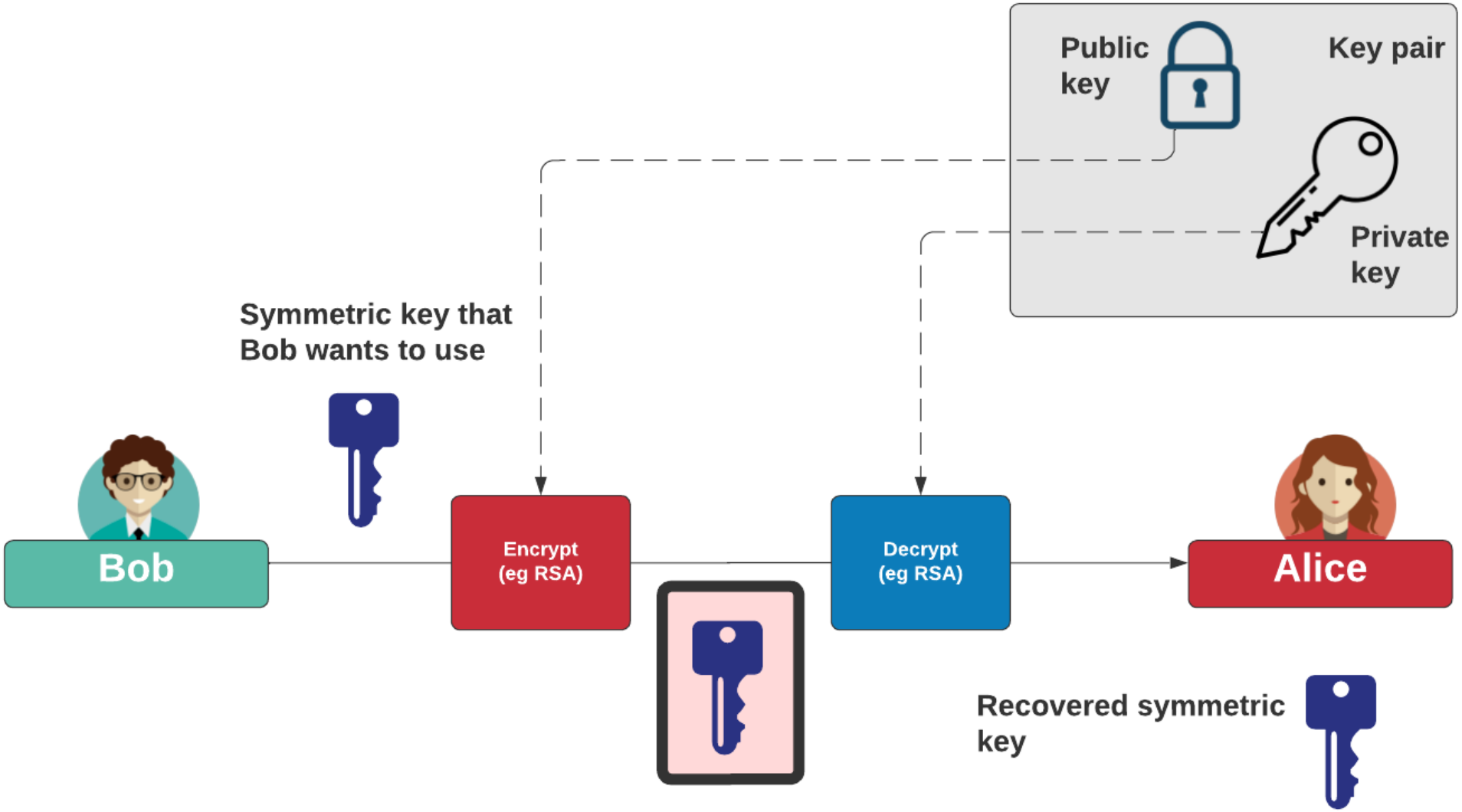
- **DH Group 5**: 1,536 bit prime.
- **DH Group 2**: 1,024 bit prime.
- **DH Group 1**: 768-bit prime.

Elliptic Curve Diffie Hellman



Common curves: **secp256r1** (P256), secp256k1 (Bitcoin) and **Curve 25519**

Key Exchange with Public Key



Key Exchange

- **Forward secrecy** (FS), which means that a comprise of the long-term keys will not compromise any previous session keys. A leakage of the public key of the server would cause all the sessions which used this specific public key to be compromised. FS thus aims to overcome this by making sure that all the sessions keys could not be compromised, even though the long-term key was compromised.
- **Ephemeral**. With some key exchange methods, the same key will be generated if the same parameters are used on either side. This can cause problems as an intruder could guess the key, or even where the key was static and never changed. With ephemeral methods, a different key is used for each connection, and, again, the leakage of any long-term would not cause all the associated session keys to be breached.

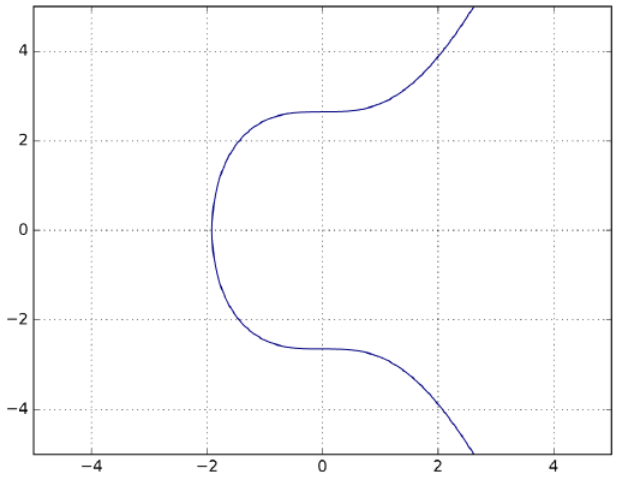
Cryptography: Digital Signatures

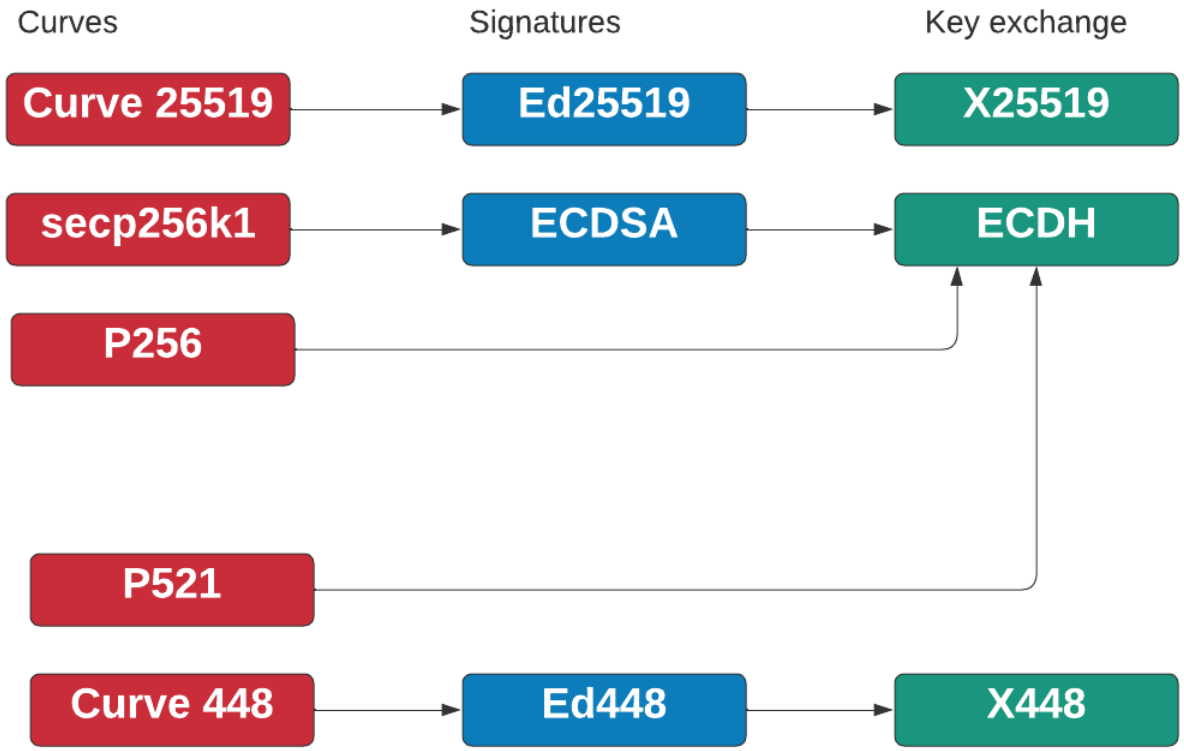
Prof Bill Buchanan OBE FRSE

https://asecuritysite.com/signatures

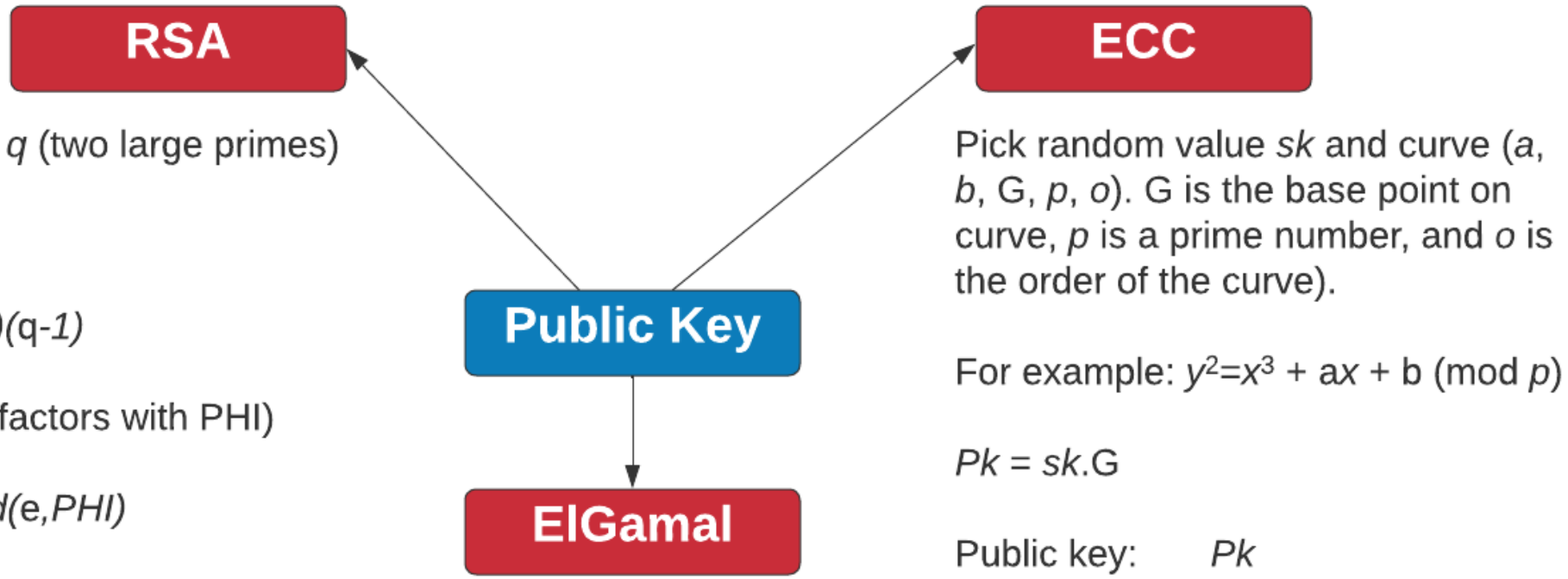
Fundamentals
Symmetric Key
Hashing
MAC
KDF
Public Key
Key Exchange
Signatures
Digital Certificates

Digital Signatures





Digital Signatures



Pick p and q (two large primes)

$N = p.q$

$PHI = (p-1)(q-1)$

Pick e (no factors with PHI)

$d = InvMod(e,PHI)$

Public: (e,N)
Private: (d,N)

$C = M^e \pmod N$
$P = C^d \pmod N$

Pick random value x, generator (g) and prime (p)

$Y = g^x \pmod p$

Public key: (Y,p)
Private: (x,p)

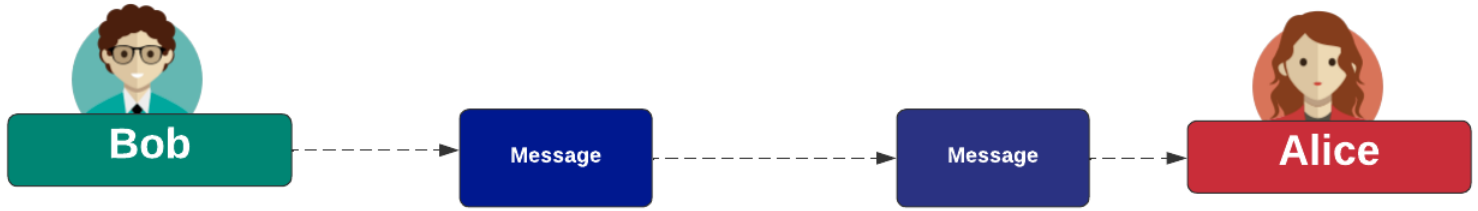
Pick random value sk and curve (a, b, G, p, o). G is the base point on curve, p is a prime number, and o is the order of the curve).

For example: $y^2=x^3 + ax + b \pmod p$

$Pk = sk.G$

Public key: Pk
Private key: sk

Digital Signatures



Hashing:
- SHA-256, SHA-3 or BLAKE. All 256-bit hashes (or greater).

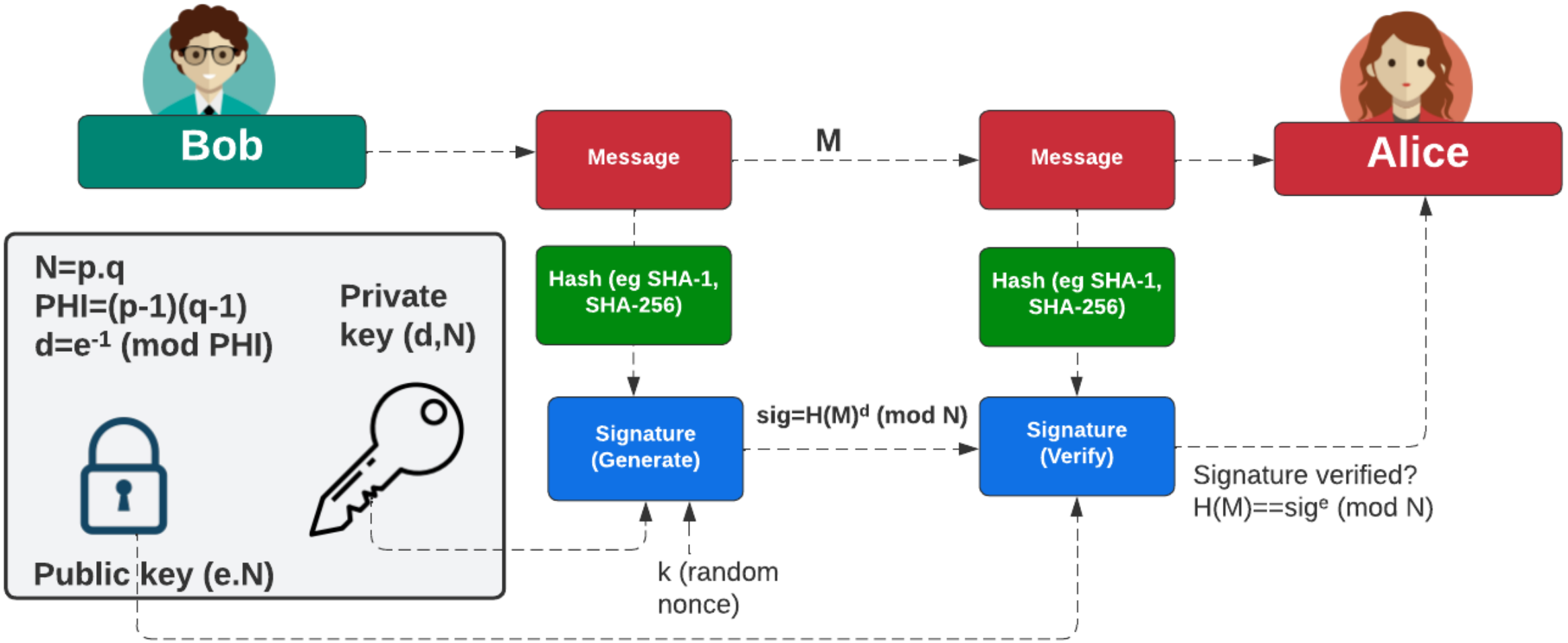
Symmetric key (Privacy and integrity):
- AES (128-bit or 256-bit). Normally with GCM mode for integrity.
- ChaCha20 (256-bit). Normally with Poly1305 for integrity.

Key exchange (Privacy and Integrity):
- ECDH (eg P256: 256-bit private key, 512-bit public key)
- X25519 (eg Curve 25519: 256-bit private key and 256-bit public key)
- RSA key exchange (no recommended, and not supported in TLS 1.3)
- Not used any more: Diffie-Hellman

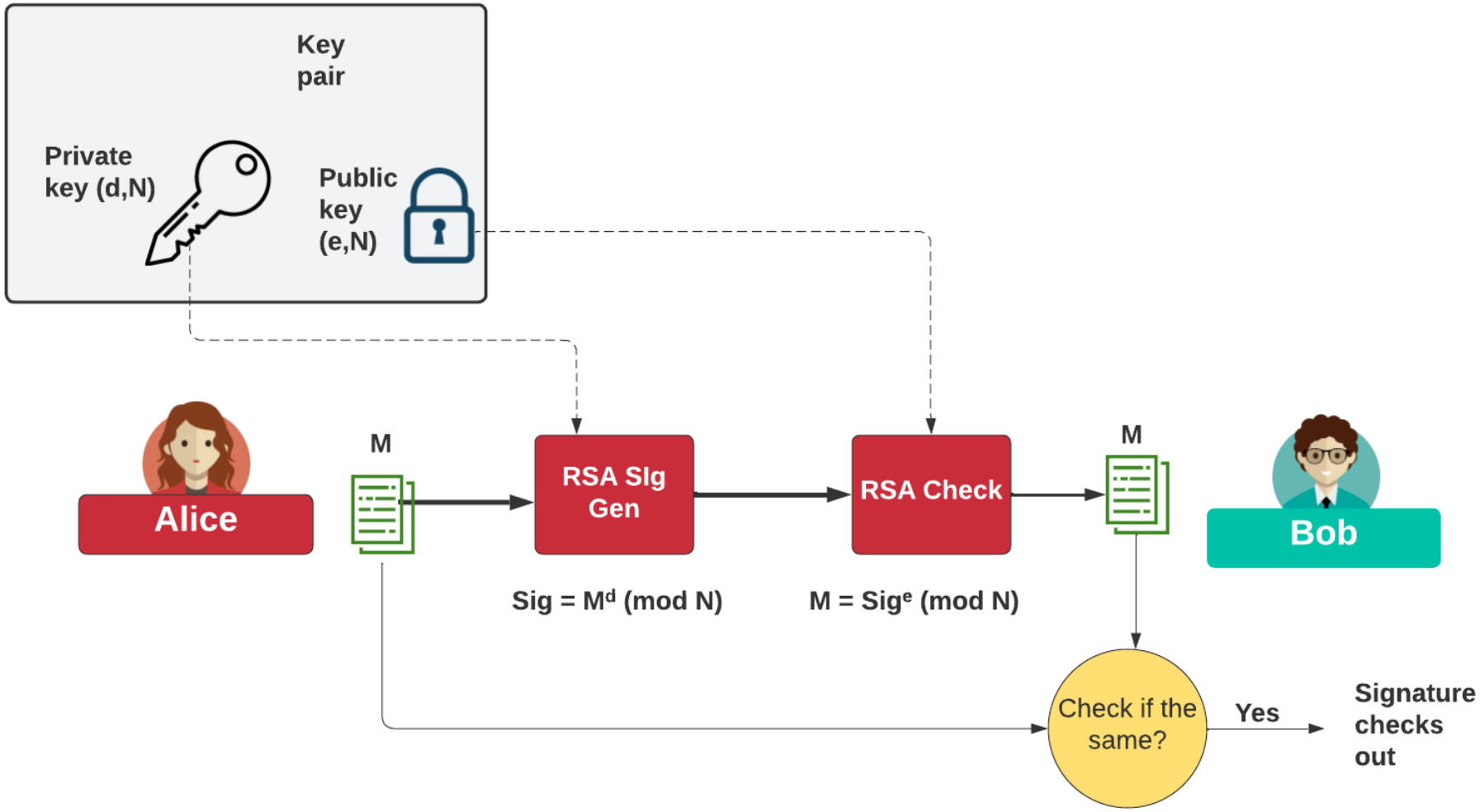
Public Key Encryption (Privacy and Integrity)
- RSA: 2,048-bit (2K) or 4,096-bit (4K) public key (Never less!)
- Not used any more: ElGamal (Discrete logs).

Digital signatures (Trust and Integrity):
- RSA: 2,048-bit (2K) or 4,096-bit (4K) public key (Never less!)
- ECDSA (eg P256: 256-bit private key, 512-bit public key)
- EdDSA (eg Ed25519: 256-bit private key, 256-bit public key).
- Not used any more: DSA (Digital Signature Algorithm).

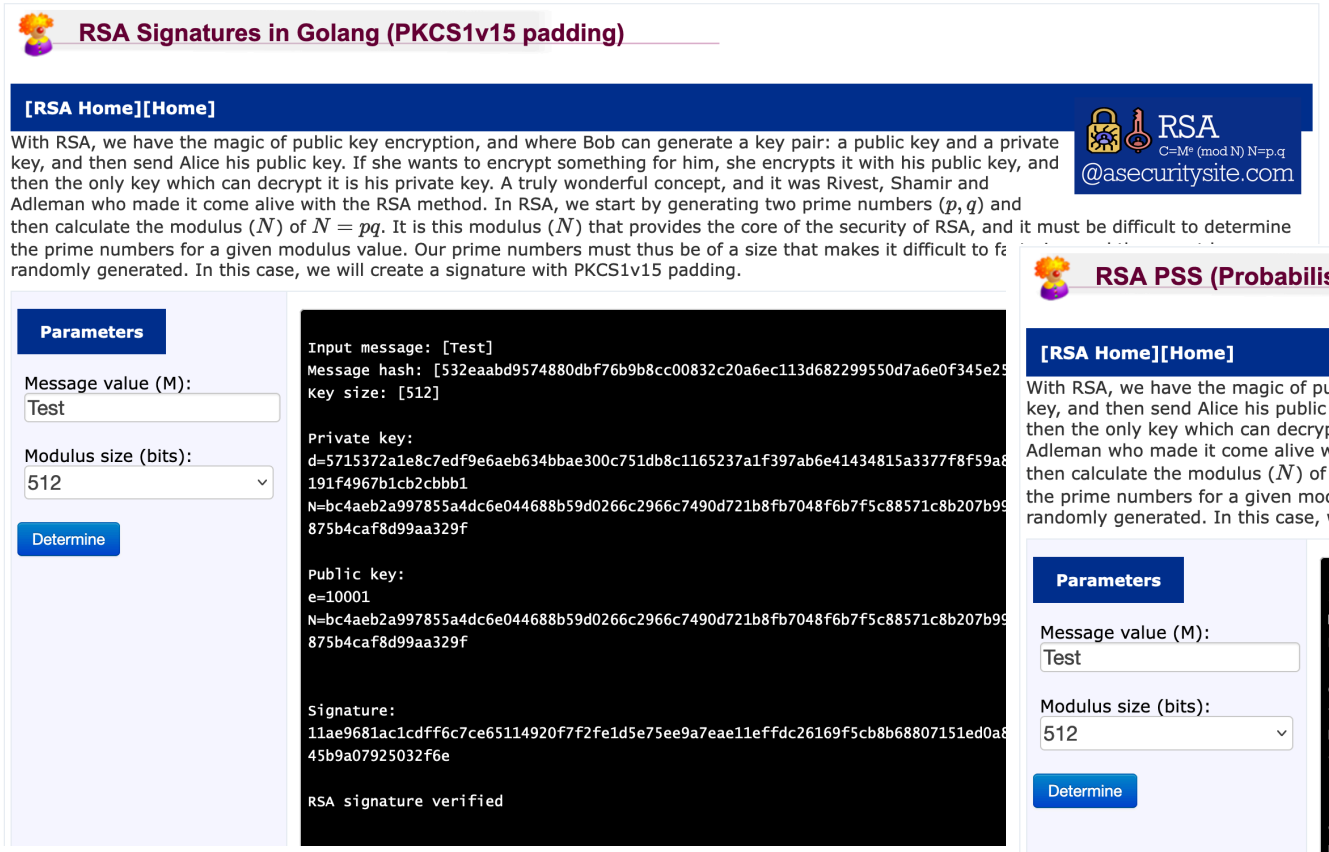
Digital Signatures

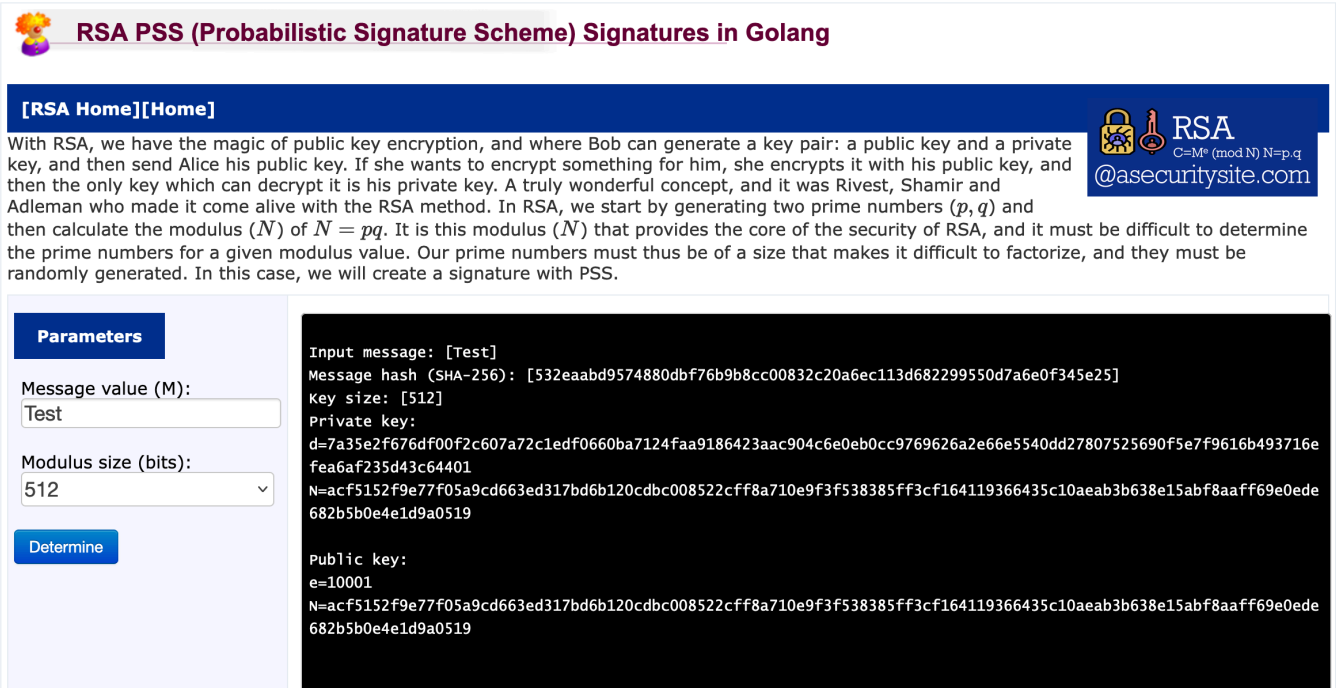


Digital Signatures - RSA



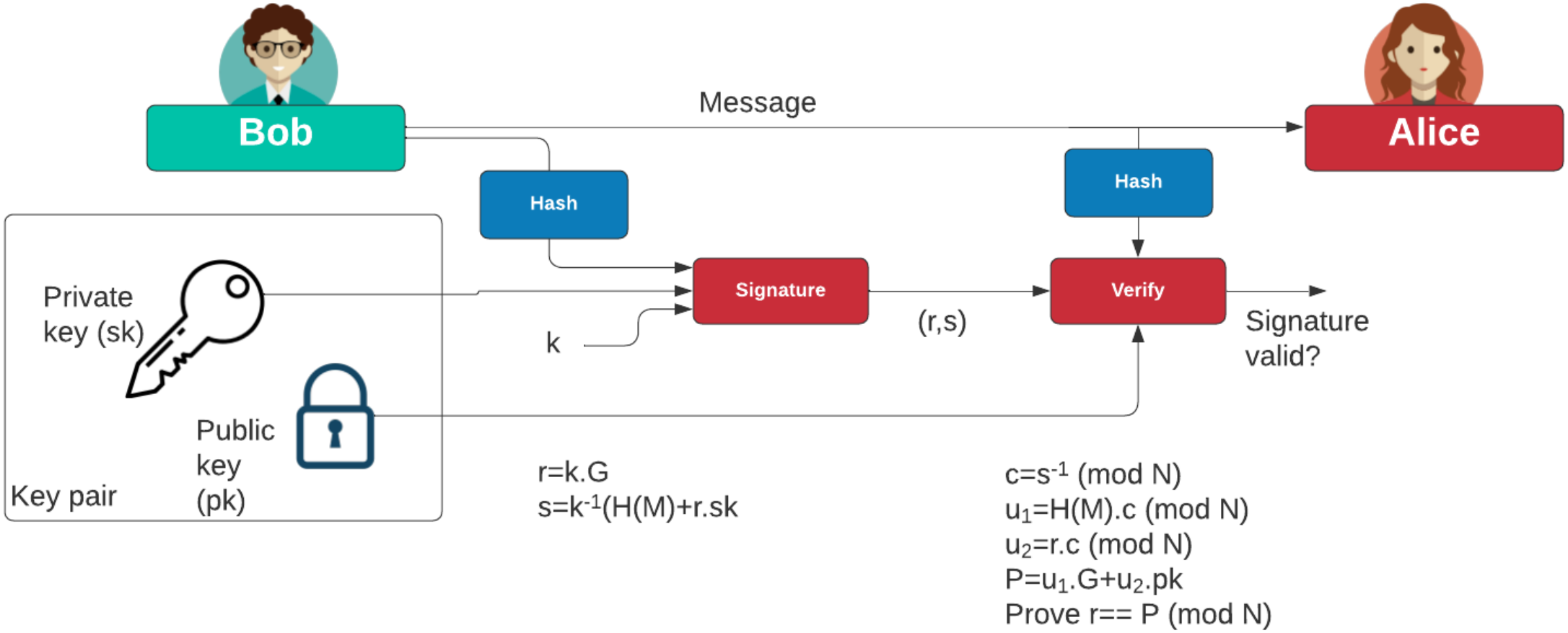
RSA Digital Signatures – PSS and PKCS1v15





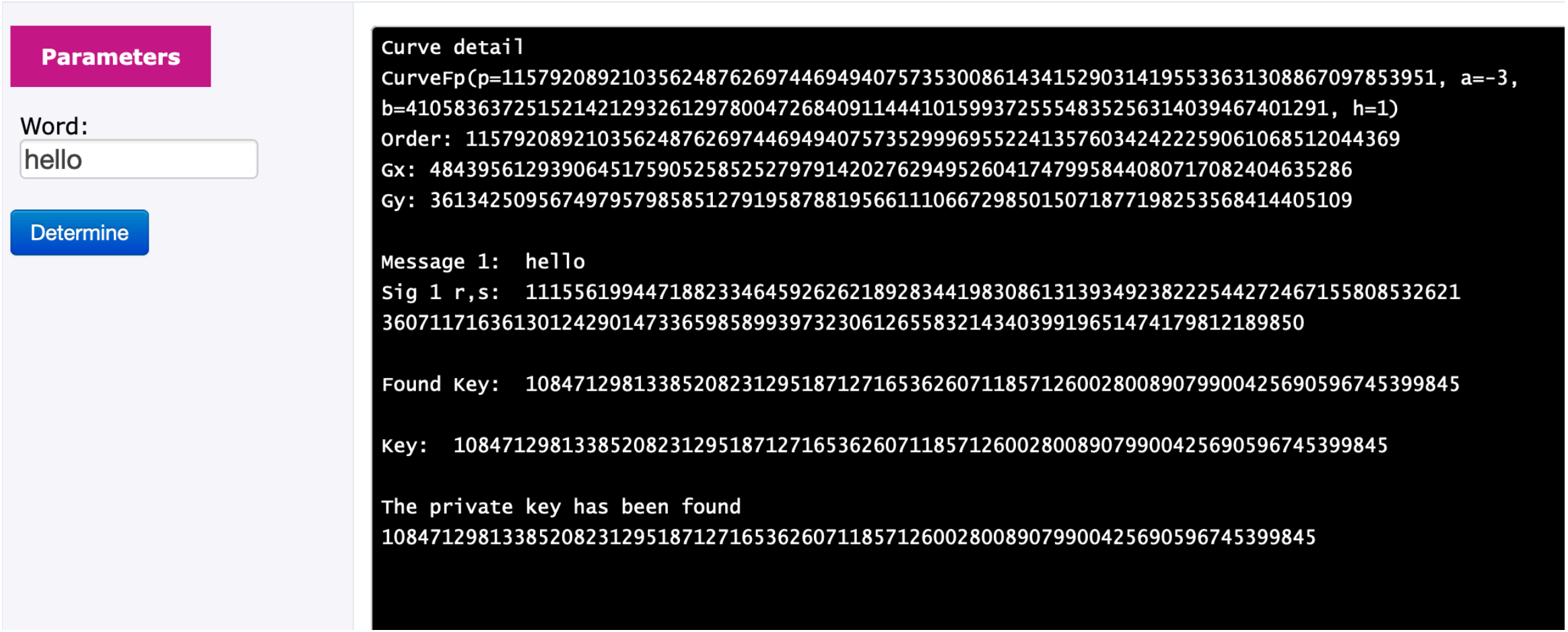
https://asecuritysite.com/rsa/go_rsa4 https://asecuritysite.com/rsa/go_rsa5

ECDSA



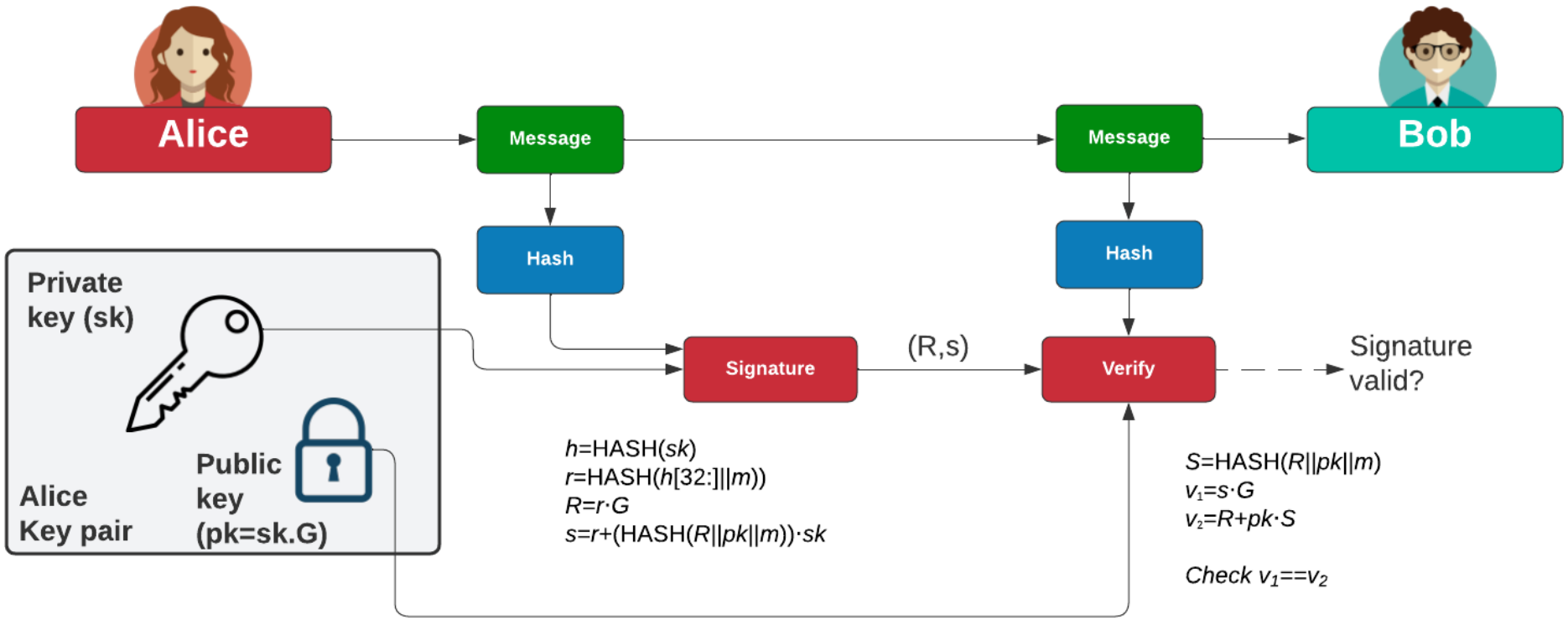
https://asecuritysite.com/ecdsa/ecdsa3

ECDSA: Leak of nonce



https://asecuritysite.com/cracking/ecd2

EdDSA



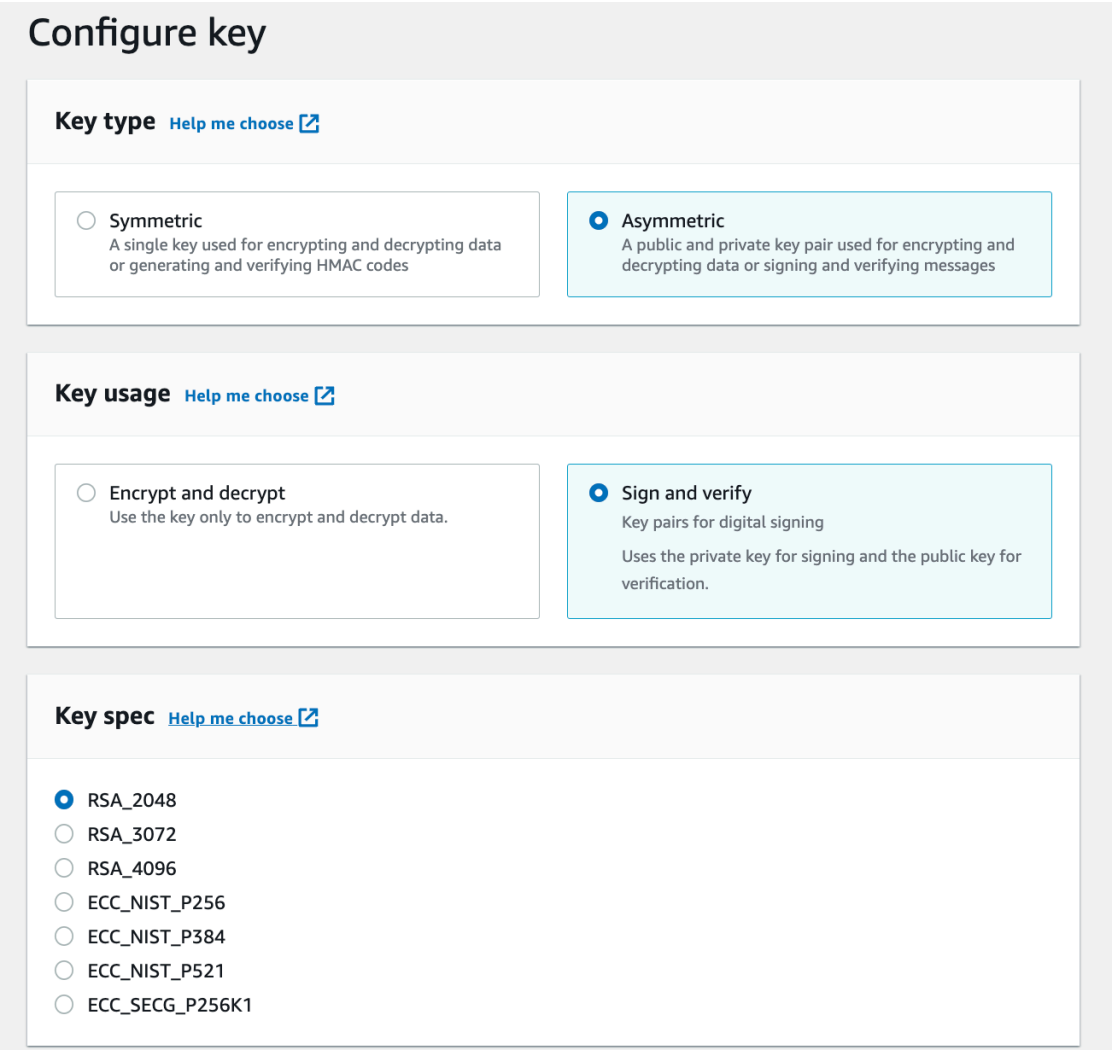
https://asecuritysite.com/golang/ecdsa_rs2

https://asecuritysite.com/golang/ecdsa_rs

BLOCKPASS IDENTITY LAB

World-leading Collaboration between Blockpass IDN and Edinburgh Napier University

Cloud Keys and Signatures



https://asecuritysite.com/aws/lab09

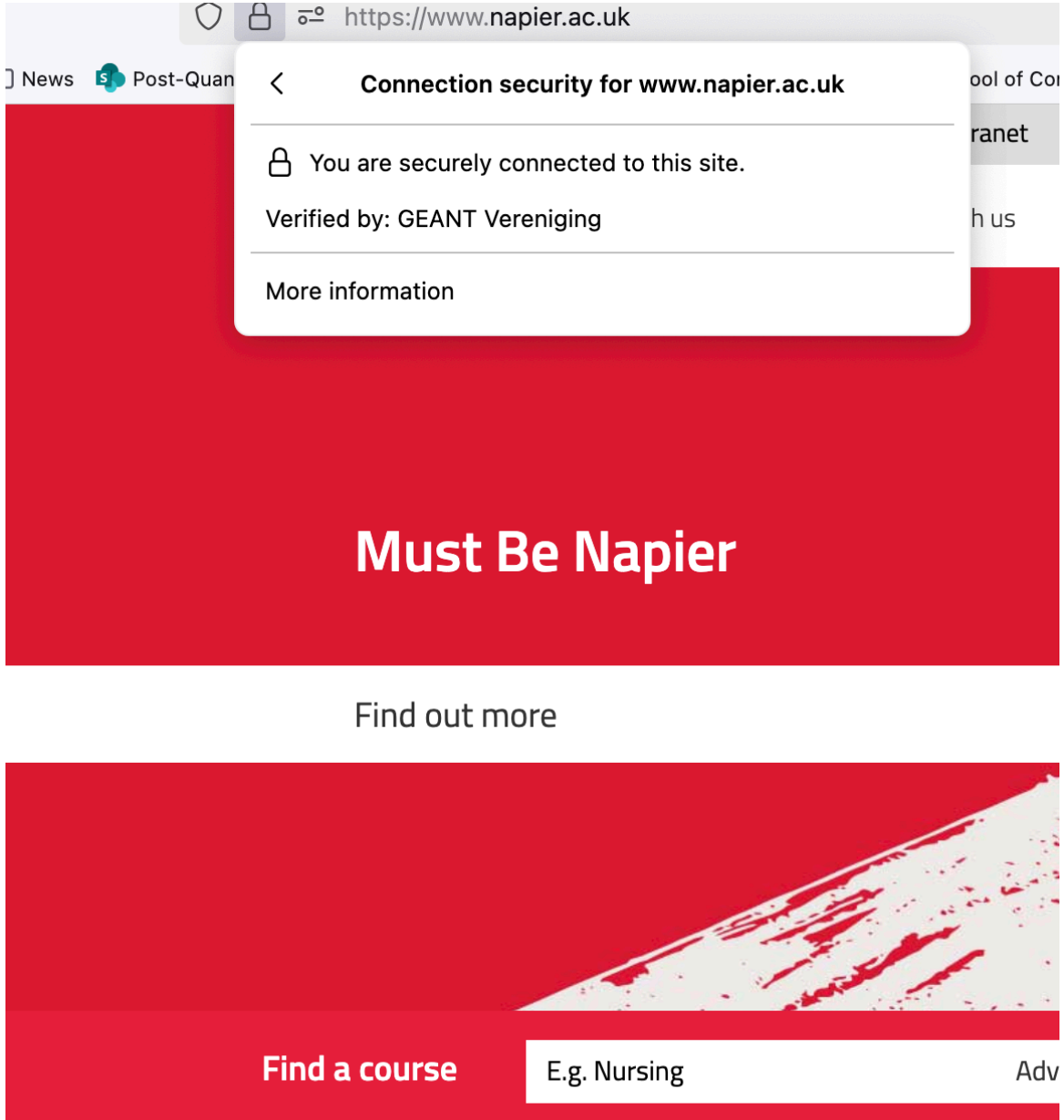
Cryptography: Digital Certificates

Prof Bill Buchanan OBE FRSE

https://asecuritysite.com/digitalcert

Fundamentals
Symmetric Key
Hashing
MAC
KDF
Public Key
Key Exchange
Signatures
Digital Certificates

Digital Cerificates



Certificate

www.napier.ac.uk | GEANT OV RSA CA 4 | USERTrust RSA Certification Authority

Subject Name

Country	GB
State/Province/County	Midlothian
Organisation	Edinburgh Napier University
Common Name	www.napier.ac.uk

Issuer Name

Country	NL
Organisation	GEANT Vereniging
Common Name	GEANT OV RSA CA 4

Validity

Not Before	Fri, 20 Sep 2024 00:00:00 GMT
Not After	Sat, 20 Sep 2025 23:59:59 GMT

Subject Alt Names

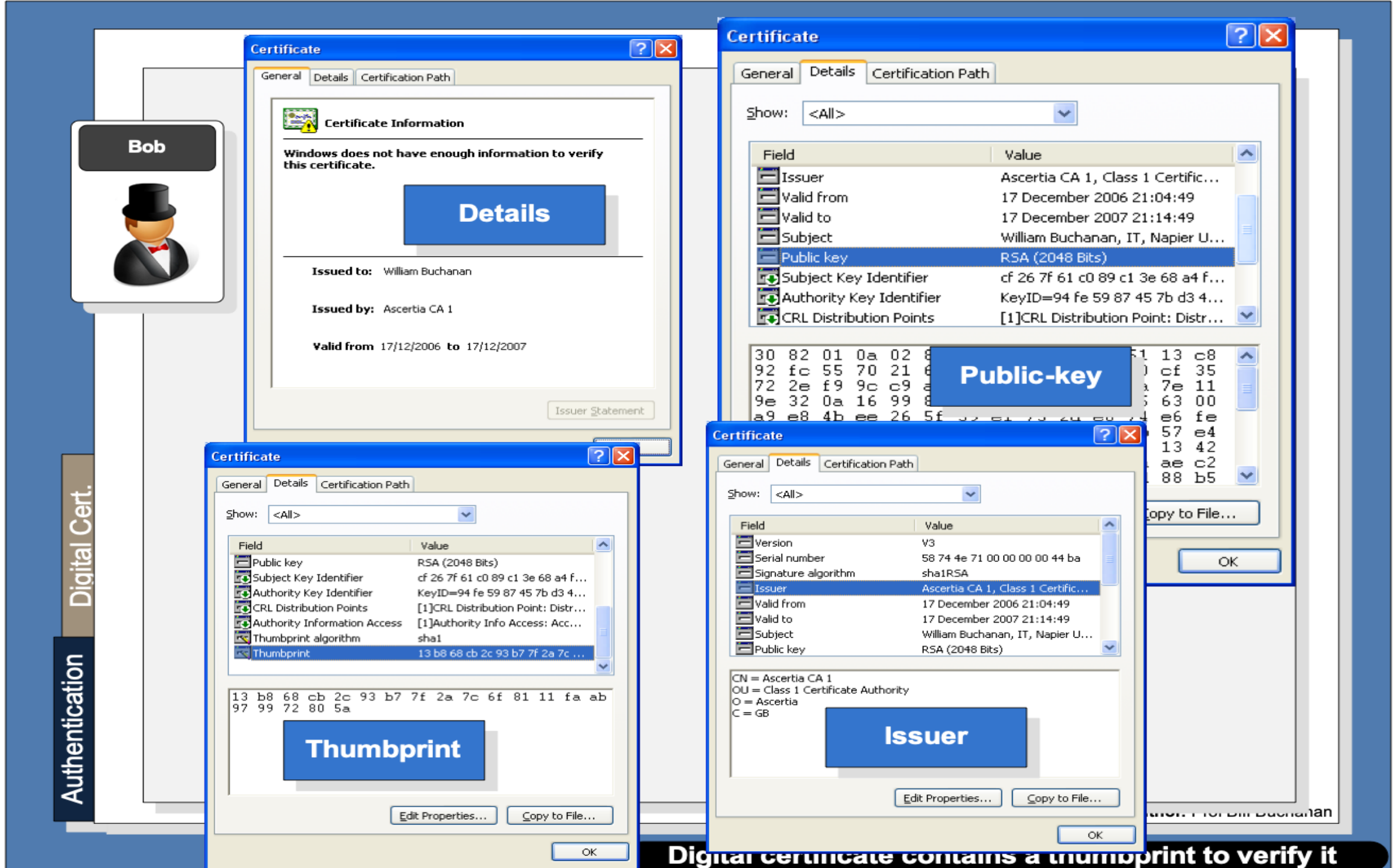
DNS Name	www.napier.ac.uk
DNS Name	napier.ac.uk
DNS Name	researchrepository.napier.ac.uk

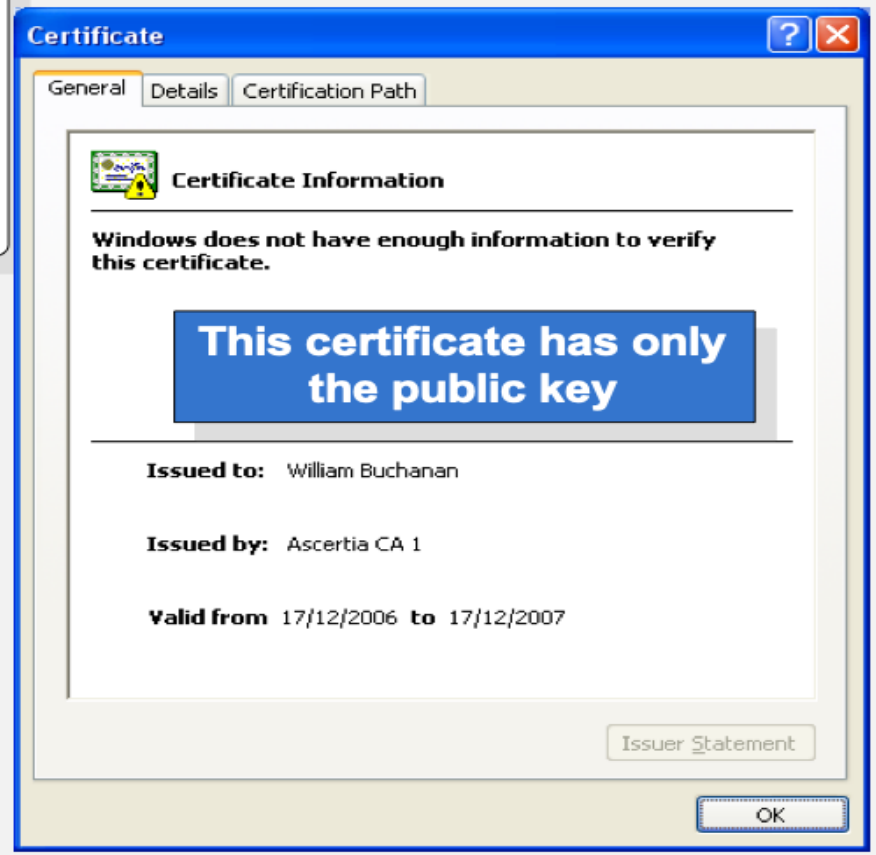
Public Key Info

Algorithm	RSA
Key Size	2048
Exponent	65537
Modulus	88:5F:4B:B0:80:5F:37:65:E5:4B:CC:18:22:13:5C:24:0F:6A:FA:65:C0:87:C...

Miscellaneous

Serial Number	00:95:C1:B6:79:0A:BC:5E:B0:C8:9E:21:0B:6B:B0:A5:3E
Signature Algorithm	SHA-384 with RSA Encryption
Version	3
Download	PEM (cert) PEM (chain)


Bob
Certificate
General
Details
Certification Path
Certificate Information
Windows does not have enough information to verify this certificate.
Details
Issued to: William Buchanan
Issued by: Ascertia CA 1
Valid from 17/12/2006 to 17/12/2007
Issuer Statement
Certificate
General
Details
Certification Path
Show: <All>
Field
Value
Issuer
Ascertia CA 1, Class 1 Certific...
Valid from
17 December 2006 21:04:49
Valid to
17 December 2007 21:14:49
Subject
William Buchanan, IT, Napier U...
Public key
RSA (2048 Bits)
Subject Key Identifier
cf 26 7f 61 c0 89 c1 3e 68 a4 f...
Authority Key Identifier
KeyID=94 fe 59 87 45 7b d3 4...
CRL Distribution Points
[1]CRL Distribution Point: Distr...
Public-key
Copy to File...
OK
Certificate
General
Details
Certification Path
Show: <All>
Field
Value
Public key
RSA (2048 Bits)
Subject Key Identifier
cf 26 7f 61 c0 89 c1 3e 68 a4 f...
Authority Key Identifier
KeyID=94 fe 59 87 45 7b d3 4...
CRL Distribution Points
[1]CRL Distribution Point: Distr...
Authority Information Access
[1]Authority Info Access: Acc...
Thumbprint algorithm
sha1
Thumbprint
13 b8 68 cb 2c 93 b7 7f 2a 7c ...
13 b8 68 cb 2c 93 b7 7f 2a 7c 6f 81 11 fa ab
97 99 72 80 5a
Thumbprint
Edit Properties...
Copy to File...
OK
Certificate
General
Details
Certification Path
Show: <All>
Field
Value
Version
V3
Serial number
58 74 4e 71 00 00 00 00 44 ba
Signature algorithm
sha1RSA
Issuer
Ascertia CA 1, Class 1 Certific...
Valid from
17 December 2006 21:04:49
Valid to
17 December 2007 21:14:49
Subject
William Buchanan, IT, Napier U...
Public key
RSA (2048 Bits)
CN = Ascertia CA 1
OU = Class 1 Certificate Authority
O = Ascertia
C = GB
Issuer
Edit Properties...
Copy to File...
OK
Digital Cert.
Authentication
Digital certificate contains a thumbprint to verify it



Digital certificates should only be distributed with the public key

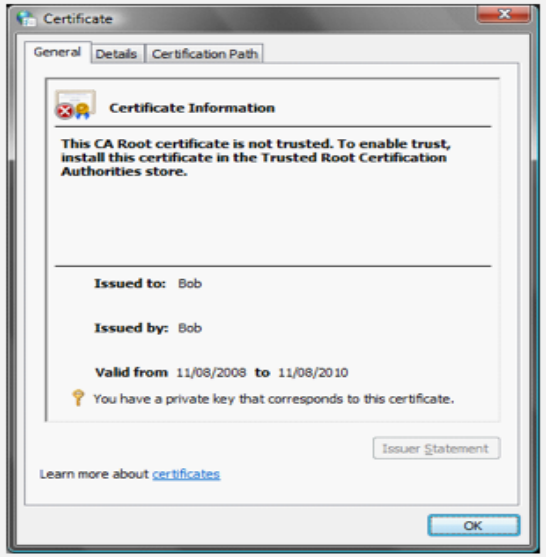
Authentication

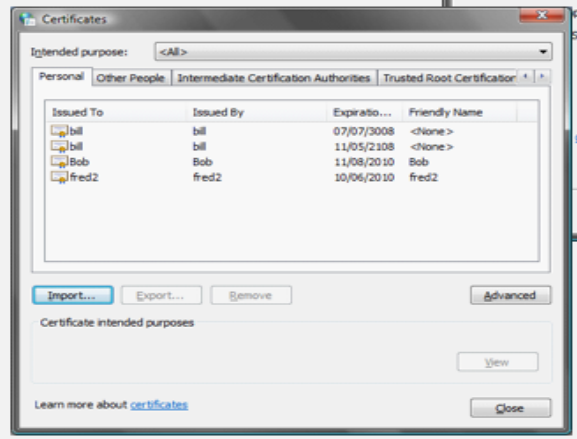
Digital Cert.

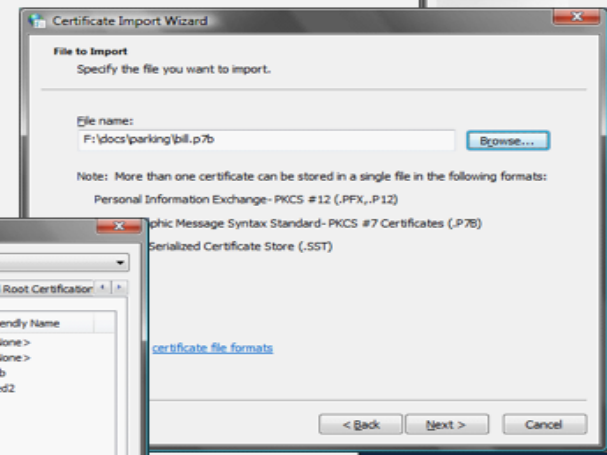
P7b format

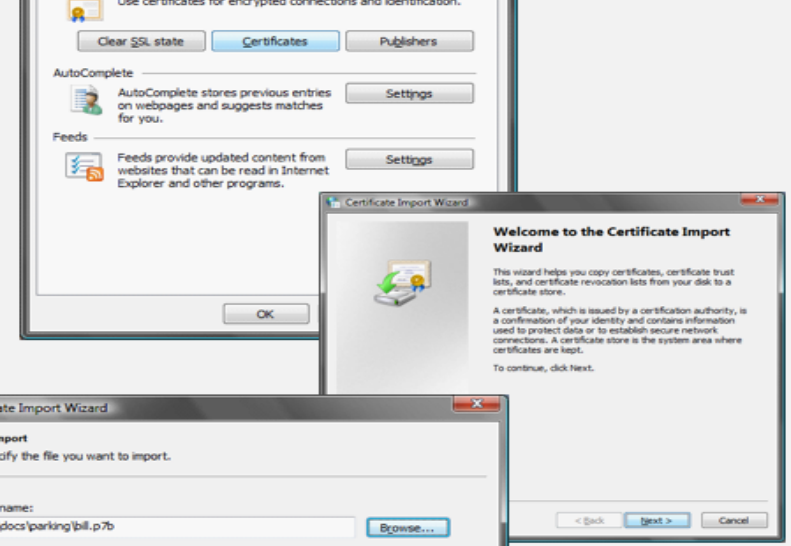
Bob

```
-----BEGIN CERTIFICATE-----
MIID2zCCA4WgAwIBAgIKWHROcQAAAABEujANBgkqhkiG 9w0BAQUFADBgMQswCQYD
VQQGEwJHQjERMA 8GA1UEChMIQXNjZXJ 0aWExJjAkBgNVBAsTHUNsYXNzIDEgQ 2Vy
dGlmaWNhdGUgQXV 0aG9yaXR 5MRYwFAYDVQQDEw 1Bc2NlcnRpYSBDQSAxMB 4XDTA2
MTIxNzIxMDQ 0OVoXDTA 3MTIxNzIxMTQ 0OVowgZ 8xJjAkBgkqhkiG 9w0BCQEWF 3cu
YnVjaGFuYW 5AbmFwaWVyLmFjLnVrMQswCQYDVQQGEwJVSzEQMA 4GA1UECBMHTG 90
aGlhbjESMBAGA 1UEBxMJRWRpbmJ 1cmdoMRowGAYDVQQKExFOYXBpZXIgVW 5pdmVy
c2l0eTELMAkGA 1UECxMCSVQxGTAXBgNVBAMTEFdpbGxpYW 0gQnVjaGFuYW 4wggEi
MA0GCSqGSIb 3DQEBAQUAA 4IBDwAwggEKAoIBAQCvCFETyJL 8VXAhbEMRzQI 0gM81
ci75nMmsOamjZcb 6fhGeMgoWmYcoscmQkrVjAKnoS +4mXznhcy 3mdOb+sZbwOvaX
M5FOxhSrV +Q86hsK 8cDc+1sqyJ 8TQtufuDNs 0NfNY 6tR6q 7CgGqQ 8/VjSxNqzK 39
iLUF1ahhyCet /ab6O/qWzL 4iVsz 2nmL 4dyAuyilhLPlVbppHGdE 6sDQXwYd 0CpfV
ZN7pauD 5fqBESfO 6bUkCieI 47AzRMQj 3kHuDt 7MexVw 7aoX+nXLP 4wn 7IamaxasF
QvhdOKyCZhYs 82JQDGatXRCqkklztmZW 5i 6GkPsE 7XvuX 265WJQ 5afhp 2hYlAgMB
AAGjggEXMIIBEzAdBgNVHQ 4EFgQUzyZ /YcCJwT 5opPHLPlcQKkOlkJwwYwYDVR 0j
BFwwWoAUlP 5Zh0V700k 6CorvRMWB 9ifVkBmhP 6Q 9MDsxCzAJBgNVBAYTAkdCMREw
DwYDVQQKEwhBc 2NlcnRpYTEZMBcGA 1UEAxMQQXNjZXJ 0aWEgUm 9vdCBDQYIBDTBN
BgNVHR 8ERjBEMEKgQKA+hjxodHRwOi 8vd3d3LmFzY 2VydGlhLmNvbS 9PbmxpbmVD
QS9jcmxzL 0FzY 2VydGlhQ 0ExL 2NsYXNzMS 5jcmwwPgYIKwYBBQUHAQEEMjAwMC 4G
CCsGAQUFBzAChiJodHRwOi 8vb2NzcC 5nbG 9iYWx 0cnVzdGZpbmRlci 5jb20vMA 0G
CSqGSIb 3DQEBBQUAA 0EATOCwGJ 1tS0kTlupmpjkMl 8IdxMmD 5WuhszjBlGsMhPxI
H+vXhL 9yaOw+Prpzy 7ajS4/3xXU 8vRANhyU 9yU 4qDA==
-----END CERTIFICATE-----
```





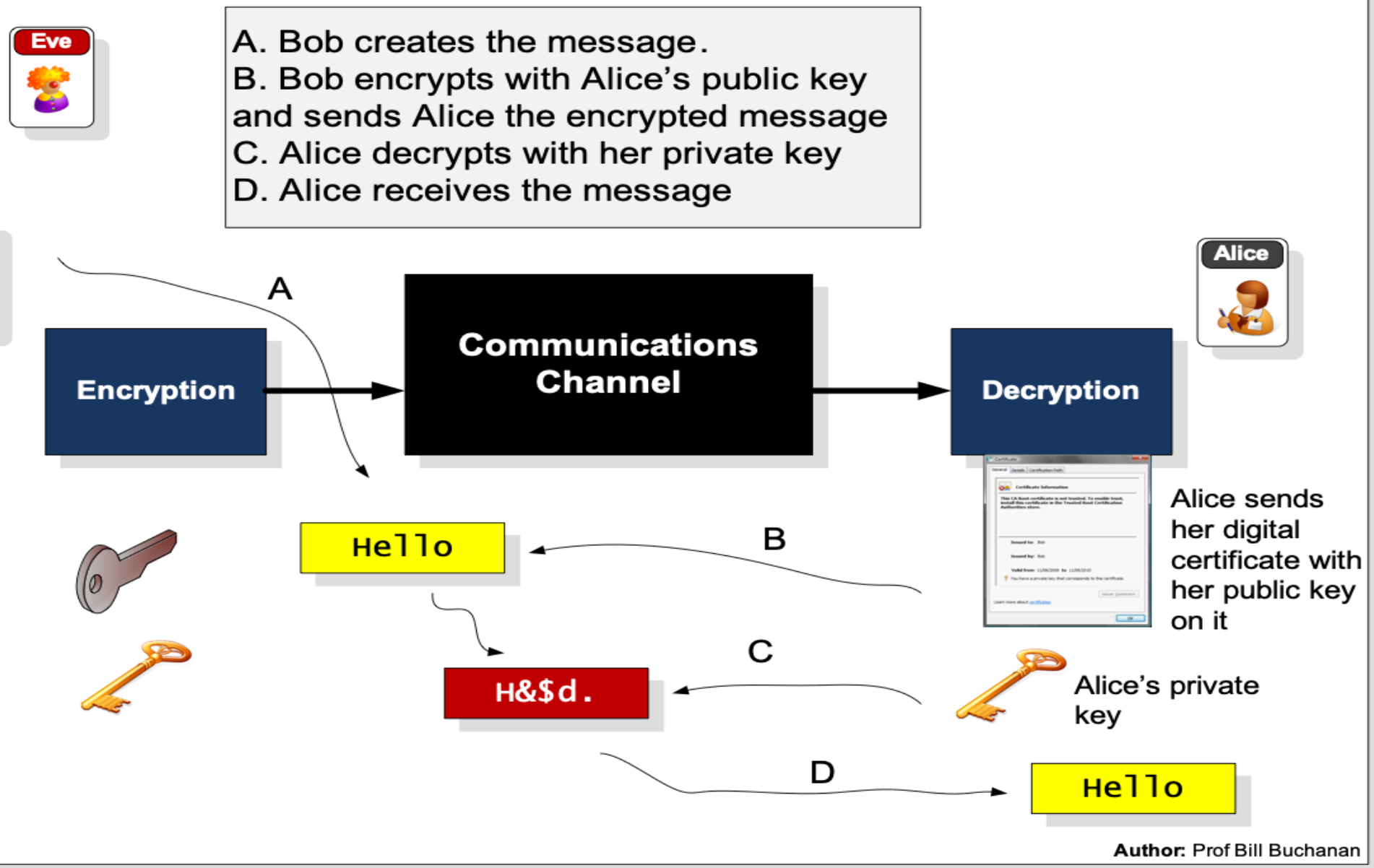


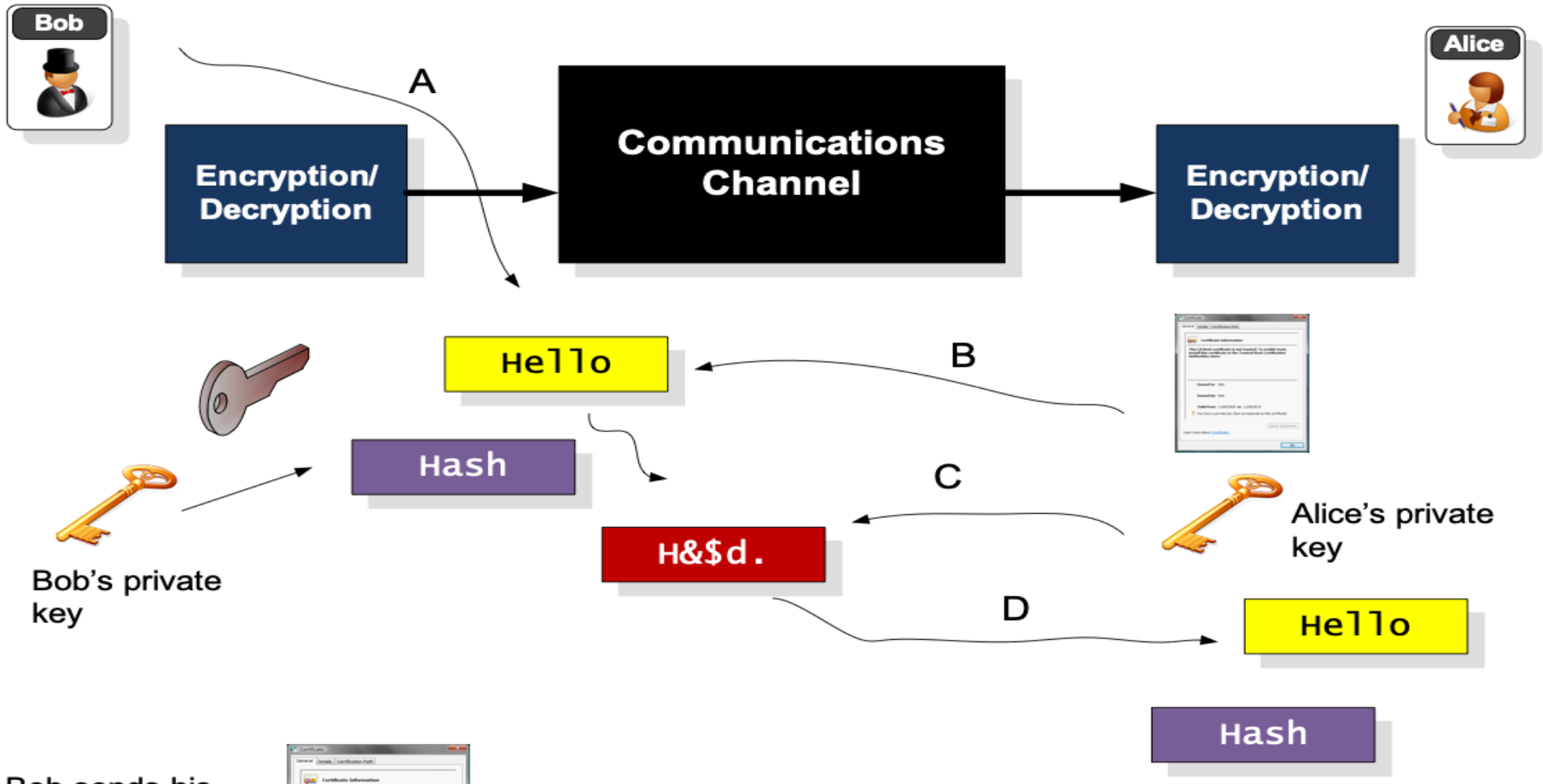


The main certificate formats include:

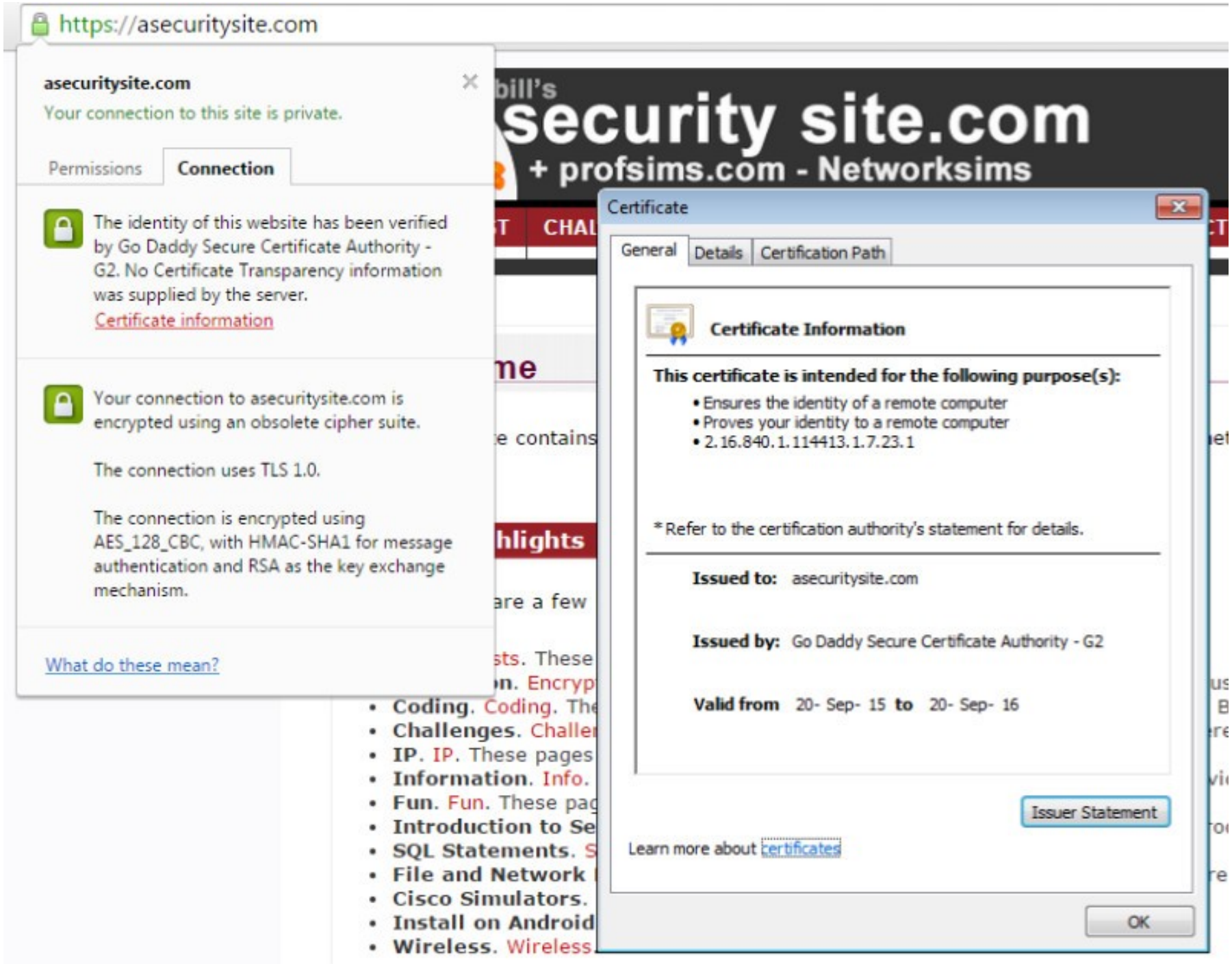
- P7b. Text format
- PFX/P12. Binary.
- SST. Binary.

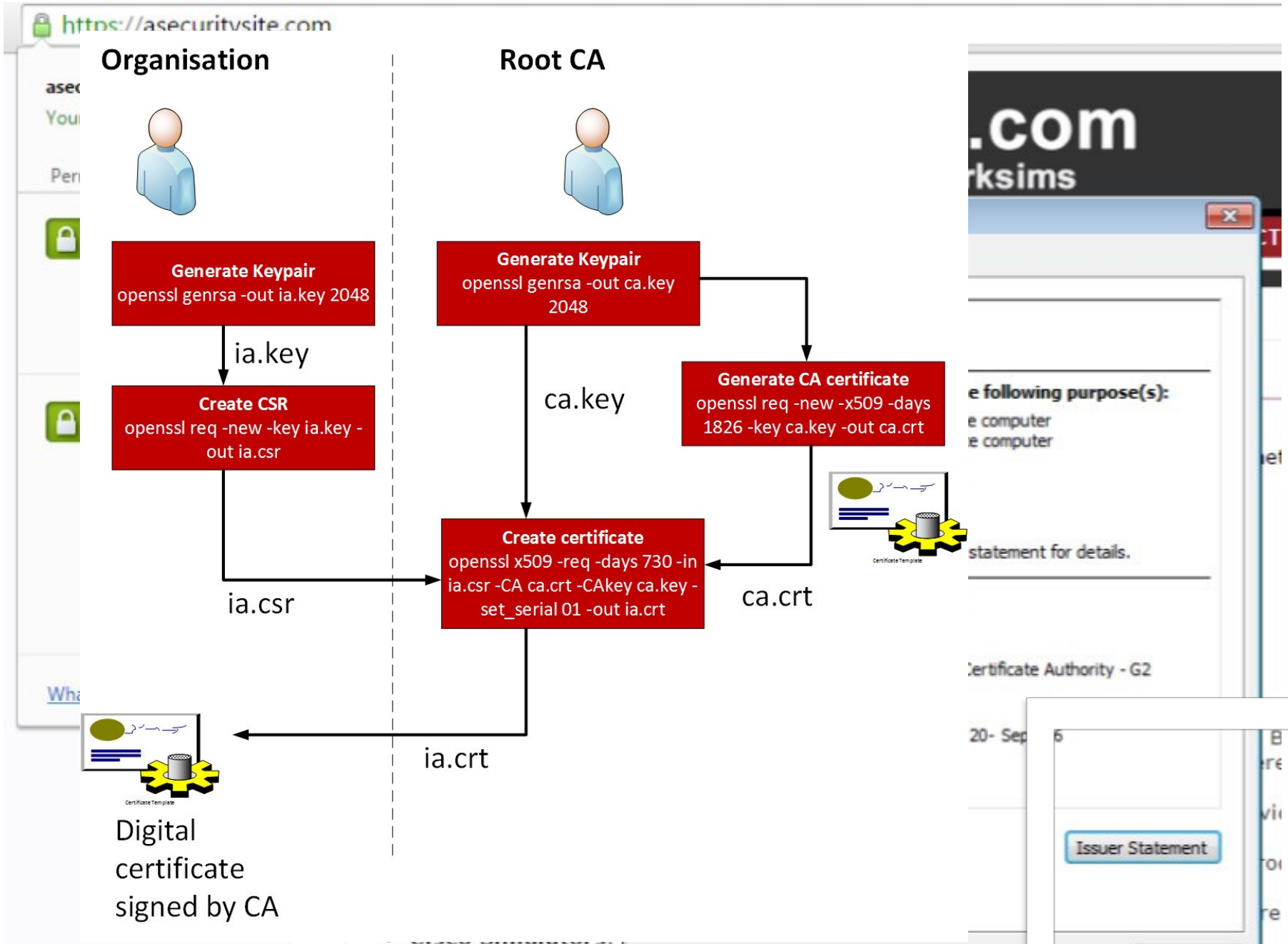
Digital certificates should only be distributed with the public key


Eve
A. Bob creates the message.
B. Bob encrypts with Alice's public key and sends Alice the encrypted message
C. Alice decrypts with her private key
D. Alice receives the message
Bob
Alice
A
Encryption
Communications Channel
Decryption
Hello
B
Alice sends her digital certificate with her public key on it
H&$d.
C
Alice's private key
D
Hello
Author: Prof Bill Buchanan

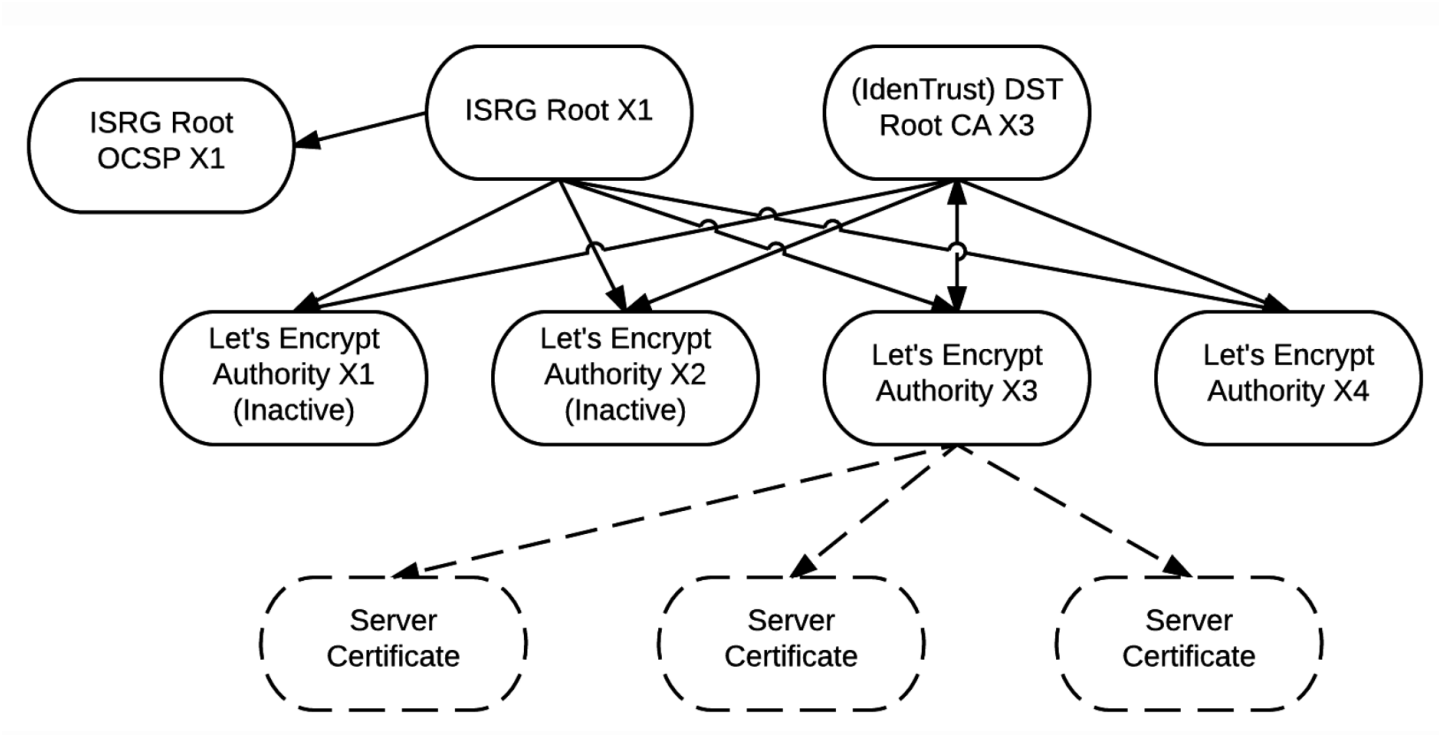


Author: Prof Bill Buchanan


https://asecuritysite.com
asecuritysite.com
Your connection to this site is private.
Permissions
Connection
The identity of this website has been verified by Go Daddy Secure Certificate Authority - G2. No Certificate Transparency information was supplied by the server.
Certificate information
Your connection to asecuritysite.com is encrypted using an obsolete cipher suite.
The connection uses TLS 1.0.
The connection is encrypted using AES_128_CBC, with HMAC-SHA1 for message authentication and RSA as the key exchange mechanism.
What do these mean?
bill's
security site.com
+ profsims.com - Networksims
Certificate
General
Details
Certification Path
Certificate Information
This certificate is intended for the following purpose(s):
• Ensures the identity of a remote computer
• Proves your identity to a remote computer
• 2.16.840.1.114413.1.7.23.1
* Refer to the certification authority's statement for details.
Issued to: asecuritysite.com
Issued by: Go Daddy Secure Certificate Authority - G2
Valid from 20- Sep- 15 to 20- Sep- 16
Issuer Statement
Learn more about certificates
OK


Organisation
Root CA
Generate Keypair
openssl genrsa -out ia.key 2048
ia.key
Create CSR
openssl req -new -key ia.key -out ia.csr
ia.csr
Generate Keypair
openssl genrsa -out ca.key 2048
ca.key
Generate CA certificate
openssl req -new -x509 -days 1826 -key ca.key -out ca.crt
ca.crt
Create certificate
openssl x509 -req -days 730 -in ia.csr -CA ca.crt -CAkey ca.key -set_serial 01 -out ia.crt
ia.crt
Digital certificate signed by CA
Issuer Statement
OK


SECURE THE WEB
Let's Encrypt


ISRG Root OCSP X1
ISRG Root X1
(IdenTrust) DST Root CA X3
Let's Encrypt Authority X1 (Inactive)
Let's Encrypt Authority X2 (Inactive)
Let's Encrypt Authority X3
Let's Encrypt Authority X4
Server Certificate
Server Certificate
Server Certificate

PEM

```
-----BEGIN CERTIFICATE-----
MIIEwTCCA6mgAwIBAgIRAO93GGFLf                    YJKoZIhvcNAQELBQAw
QjELMAkGA1UEBhMCVVMxHjAcBgNVBAoTFUdvb2dsZSBUcnVzdCBTZXJ2aWNlczET
MBEGA1UEAxMKR1RTIENBIDFPMTAeFw0yMDAyMTIxMTQ3NDFaFw0yMDA1MDYxMTQ3
NDFaMGgxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQH
Ew1Nb3VudGFpbiBWaWV3MRMwEQYDVQQKEwpHb29nbGUgTExDMRcwFQYDVQQDEw53
d3cuZ29vZ2xlLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCzzLJYFJb8W
TpQxWL0TySDvfEWCkMe7l81CFspT60knl13YILNdTM22sUwPycyogKjBSaQZ9Axi
hCeGEAiJkbejggJVMIICUTAOBgNVHQ8BAf8EBAMCB4AwEwYDVR0lBAwwCgYIKwYB
BQUHAwEwDAYDVR0TAQH/BAIwADAdBgNVHQ4EFgQU9Ty3t90o7UW9+Hc6kv/j9511
BBwwHwYDVR0jBBgwFoAUmNH4bhDrz5vsYJ8YkBug630J/SswZAYIKwYBBQUHAQEE
WDBWMCcGCCsGAQUFBzABhhtodHRwOi8vb2NzcC5wa2kuZ29vZy9ndHMxbzEwKwYI
KwYBBQUHMAKGH2h0dHA6Ly9wa2kuZ29vZy9nc3IyL0dUUzFPMS5jcnQwGQYDVR0R
BBIwEIIOd3d3Lmdvb2dsZS5jb20wIQYDVR0gBBowGDAIBgZngQwBAgIwDAYKKwYB
BAHWeQIFAzAvBgNVHR8EKDAmMCSgIqAghh5odHRwOi8vY3JsLnBraS5nb29nL0dU
UzFPMS5jcmwwggEFBgorBgEEAdZ5AgQCBIH2BIHzAPEAdgCyHgXMi6LNiiBOh2b5
K7mKJSBna9r6cOeySVMt74uQXgAAAXA5cNqwAAAEAwBHMEUCIQCojKtzOe8l1JYK
HmKnbt1puqT4AE3peMVAVk/WeWGp1QIgBvRMbNkwF6i8+JVv3CfTHKvFd8e00+GY
PbXJZb1drKEAdwBep3P531bA57U2SH3QSeAyepGaDIShEhKEGHWWgXFFWAAAAXA5
cNrLAAAEAwBIMEYCIQCQSONppELPODh65oFT5ZAGsSQz4FsjNNZedgS7WqzLnQIh
AMAru0QUuoE3RWngfhI6W7n1kxkEmd0vSZTe6+Ql+JVuMA0GCSqGSIb3DQEBCwUA
A4IBAQCDU8CVusGstVk1lAmrtg3DyYhNV1UnKyLk3RrHKumwYz5mC25bWGoLVKvv
pwxBN3zA329/9JZq0mnn0vmPMxjTDvh9sVQ7g4znwha/KJfzL8AZKUdldD9O+hDO
tlHtqVqITZPCAI/ANvbHdZiMEePB8eA+oTiw2ucChqLlsop5Mio8ckg7aXG4/QfC
AIDbvkoffK44rs4UEpsaqGe9QqMJjTuO7ZCZrFd1m3geq2ARKPHgPoPM6UbDdqg0
TNQo9C0F5ZlOkOgV/qshvpTO4YzE7TB2U571eYEqNXH48syVx8XSk3P7FjM7FIZ2
IzbJSEipZJm8DsP10fFXpToIn+zK
-----END CERTIFICATE-----
```

PKCS#7

```
-----BEGIN PKCS7-----
MIIE8gYJKoZIhvcNAQcCoIIE4zCCBN8CAQExADALBgkqhkiG9w0BBwGgggTFMIIE
wTCCA6mgAwIBAgIRAO93GGFLfHwOCAAAAAAucZgwDQYJKoZIhvcNAQELBQAwQjEL
MAkGA1UEBhMCVVMxHjAcBgNVBAoTFUdvb2dsZSBUcnVzdCBTZXJ2aWNlczETMBEG
A1UEAxMKR1RTIENBIDFPMTAeFw0yMDAyMTIxMTQ3NDFaFw0yMDA1MDYxMTQ3NDFa
MGgxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQHEw1N
b3VudGFpbiBWaWV3MRMwEQYDVQQKEwpHb29nbGUgTExDMRcwFQYDVQQDEw53d3cu
Z29vZ2xlLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCzzLJYFJb8WTpQx
WL0TySDvfEWCkMe7l81CFspT60knl13YILNdTM22sUwPycyogKjBSaQZ9AxihCeG
EAiJkbejggJVMIICUTAOBgNVHQ8BAf8EBAMCB4AwEwYDVR0lBAwwCgYIKwYBBQUH
AwEwDAYDVR0TAQH/BAIwADAdBgNVHQ4EFgQU9Ty3t90o7UW9+Hc6kv/j9511BBww
HwYDVR0jBBgwFoAUmNH4bhDrz5vsYJ8YkBug630J/SswZAYIKwYBBQUHAQEEWDBW
MCcGCCsGAQUFBzABhhtodHRwOi8vb2NzcC5wa2kuZ29vZy9ndHMxbzEwKwYIKwYB
BQUHMAKGH2h0dHA6Ly9wa2kuZ29vZy9nc3IyL0dUUzFPMS5jcnQwGQYDVR0RBBIw
EIIOd3d3Lmdvb2dsZS5jb20wIQYDVR0gBBowGDAIBgZngQwBAgIwDAYKKwYBBAHW
eQIFAzAvBgNVHR8EKDAmMCSgIqAghh5odHRwOi8vY3JsLnBraS5nb29nL0dUUzFP
MS5jcmwwggEFBgorBgEEAdZ5AgQCBIH2BIHzAPEAdgCyHgXMi6LNiiBOh2b5K7mK
JSBna9r6cOeySVMt74uQXgAAAXA5cNqwAAAEAwBHMEUCIQCojKtzOe8l1JYKHmKn
bt1puqT4AE3peMVAVk/WeWGp1QIgBvRMbNkwF6i8+JVv3CfTHKvFd8e00+GYPbXJ
Zb1drKEAdwBep3P531bA57U2SH3QSeAyepGaDIShEhKEGHWWgXFFWAAAAXA5cNrL
AAAEAwBIMEYCIQCQSONppELPODh65oFT5ZAGsSQz4FsjNNZedgS7WqzLnQIhAMAr
u0QUuoE3RWngfhI6W7n1kxkEmd0vSZTe6+Ql+JVuMA0GCSqGSIb3DQEBCwUAA4IB
AQCDU8CVusGstVk1lAmrtg3DyYhNV1UnKyLk3RrHKumwYz5mC25bWGoLVKvvpwxB
N3zA329/9JZq0mnn0vmPMxjTDvh9sVQ7g4znwha/KJfzL8AZKUdldD9O+hDOtlHt
qVqITZPCAI/ANvbHdZiMEePB8eA+oTiw2ucChqLlsop5Mio8ckg7aXG4/QfCAIDb
vkoffK44rs4UEpsaqGe9QqMJjTuO7ZCZrFd1m3geq2ARKPHgPoPM6UbDdqg0TNQo
9C0F5ZlOkOgV/qshvpTO4YzE7TB2U571eYEqNXH48syVx8XSk3P7FjM7FIZ2IzbJ
SEipZJm8DsP10fFXpToIn+zKoQAxAA==
-----END PKCS7-----
```

Base64

```
openssl x509 -outform der -in www-google-com.pem -out google.crt
openssl pkcs12 -export -in server.pem -out keystore.pkcs12
```

Binary

DER

CER

PEM

```
-----BEGIN CERTIFICATE-----
MIIEwTCCA6mgAwIBAgIRAO93GGFLf                    YJKoZIhvcNAQELBQAw
QjELMAkGA1UEBhMCVVMxHjAcBgNVBAoTFUdvb2dsZSBUcnVzdCBTZXJ2aWNlczET
MBEGA1UEAxMKR1RTIENBIDFPMTAeFw0yMDAyMTIxMTQ3NDFaFw0yMDA1MDYxMTQ3
NDFaMGgxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQH
Ew1Nb3VudGFpbiBWaWV3MRMwEQYDVQQKEwpHb29nbGUgTExDMRcwFQYDVQQDEw53
d3cuZ29vZ2xlLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCzzLJYFJb8W
TpQxWL0TySDvfEWCkMe7l81CFspT60knl13YILNdTM22sUwPycyogKjBSaQZ9Axi
[illegible]
AIDbvkoffK44rs4UEpsaqGe9QqMJjTuO7ZCZrFd1m3geq2ARKPHgPoPM6UbDdqg0
TNQo9C0F5ZlOkOgV/qshvpTO4YzE7TB2U571eYEqNXH48syVx8XSk3P7FjM7FIZ2
IzbJSEipZJm8DsP10fFXpToIn+zK
-----END CERTIFICATE-----
```

- X.509 Certificate (DER)
- X.509 Certificate (PKCS#7)

PKCS#1 v2 - Padding for Public Key

PKCS #7 v1.5 - Cryptography Message Syntax

PKCS 10 v1.7 - Certificate Request Standard.

PKCS#7

```
-----BEGIN PKCS7-----
MIIE8gYJKoZIhvcNAQcCoIIE4zCCBN8CAQExADALBgkqhkiG9w0BBwGgggTFMIIE
wTCCA6mgAwIBAgIRAO93GGFLfHwOCAAAAAAucZgwDQYJKoZIhvcNAQELBQAwQjEL
MAkGA1UEBhMCVVMxHjAcBgNVBAoTFUdvb2dsZSBUcnVzdCBTZXJ2aWNlczETMBEG
A1UEAxMKR1RTIENBIDFPMTAeFw0yMDAyMTIxMTQ3NDFaFw0yMDA1MDYxMTQ3NDFa
MGgxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYwFAYDVQQHEw1N
b3VudGFpbiBWaWV3MRMwEQYDVQQKEwpHb29nbGUgTExDMRcwFQYDVQQDEw53d3cu
Z29vZ2xlLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCzzLJYFJb8WTpQx
WL0TySDvfEWCkMe7l81CFspT60knl13YILNdTM22sUwPycyogKjBSaQZ9AxihCeG
EAiJkbejggJVMIICUTAOBgNVHQ8BAf8EBAMCB4AwEwYDVR0lBAwwCgYIKwYBBQUH
AwEwDAYDVR0TAQH/BAIwADAdBgNVHQ4EFgQU9Ty3t90o7UW9+Hc6kv/j9511BBww
HwYDVR0jBBgwFoAUmNH4bhDrz5vsYJ8YkBug630J/SswZAYIKwYBBQUHAQEEWDBW
MCcGCCsGAQUFBzABhhtodHRwOi8vb2NzcC5wa2kuZ29vZy9ndHMxbzEwKwYIKwYB
BQUHMAKGH2h0dHA6Ly9wa2kuZ29vZy9nc3IyL0dUUzFPMS5jcnQwGQYDVR0RBBIw
EIIOd3d3Lmdvb2dsZS5jb20wIQYDVR0gBBowGDAIBgZngQwBAgIwDAYKKwYBBAHW
eQIFAzAvBgNVHR8EKDAmMCSgIqAghh5odHRwOi8vY3JsLnBraS5nb29nL0dUUzFP
MS5jcmwwggEFBgorBgEEAdZ5AgQCBIH2BIHzAPEAdgCyHgXMi6LNiiBOh2b5K7mK
JSBna9r6cOeySVMt74uQXgAAAXA5cNqwAAAEAwBHMEUCIQCojKtzOe8l1JYKHmKn
bt1puqT4AE3peMVAVk/WeWGp1QIgBvRMbNkwF6i8+JVv3CfTHKvFd8e00+GYPbXJ
Zb1drKEAdwBep3P531bA57U2SH3QSeAyepGaDIShEhKEGHWWgXFFWAAAAXA5cNrL
AAAEAwBIMEYCIQCQSONppELPODh65oFT5ZAGsSQz4FsjNNZedgS7WqzLnQIhAMAr
u0QUuoE3RWngfhI6W7n1kxkEmd0vSZTe6+Ql+JVuMA0GCSqGSIb3DQEBCwUAA4IB
AQCDU8CVusGstVk1lAmrtg3DyYhNV1UnKyLk3RrHKumwYz5mC25bWGoLVKvvpwxB
N3zA329/9JZq0mnn0vmPMxjTDvh9sVQ7g4znwha/KJfzL8AZKUdldD9O+hDOtlHt
qVqITZPCAI/ANvbHdZiMEePB8eA+oTiw2ucChqLlsop5Mio8ckg7aXG4/QfCAIDb
vkoffK44rs4UEpsaqGe9QqMJjTuO7ZCZrFd1m3geq2ARKPHgPoPM6UbDdqg0TNQo
9C0F5ZlOkOgV/qshvpTO4YzE7TB2U571eYEqNXH48syVx8XSk3P7FjM7FIZ2IzbJ
SEipZJm8DsP10fFXpToIn+zKoQAxAA==
-----END PKCS7-----
```

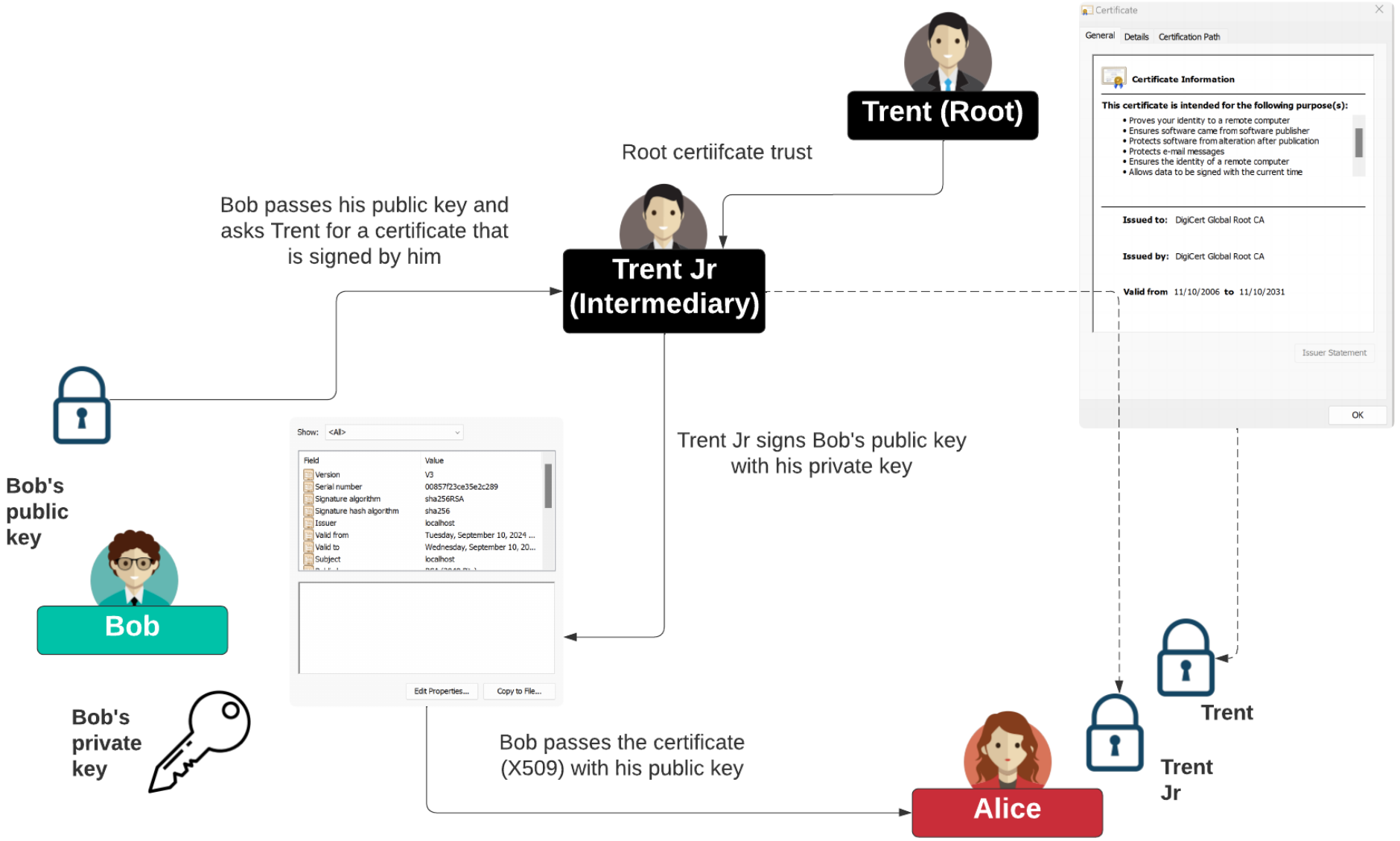
Base64

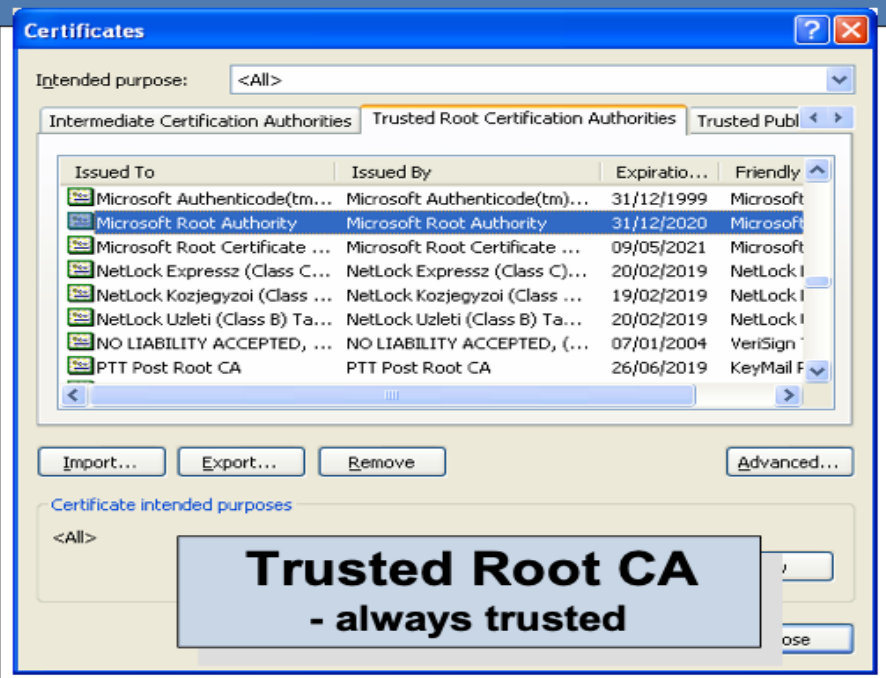
```
openssl x509 -outform der -in www-google-com.pem -out google.crt
openssl pkcs12 -export -in server.pem -out keystore.pkcs12
```

Binary

DER
CER

Digital Cerificates

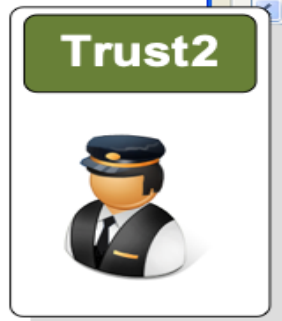


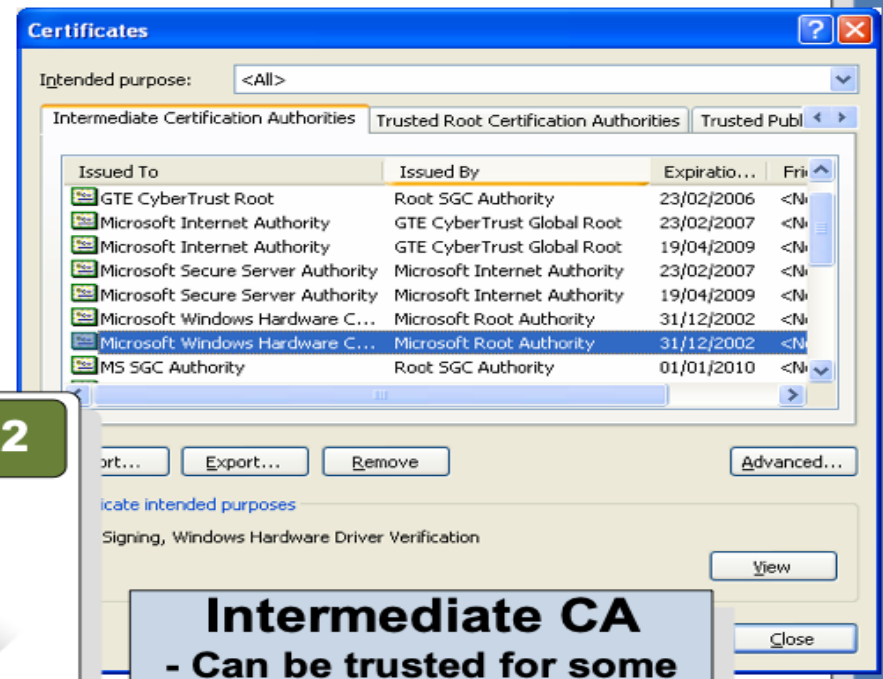


Trusted Root CA

Certificate purposes:

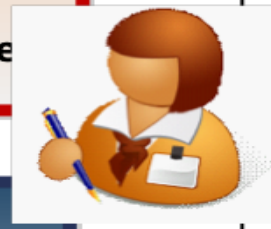
- Secure email.
- Server authentication.
- Code signing.
- Driver authentication.
- Time stamping.
- Client authentication.
- IP tunnelling.
- EFS (Encrypted File System).

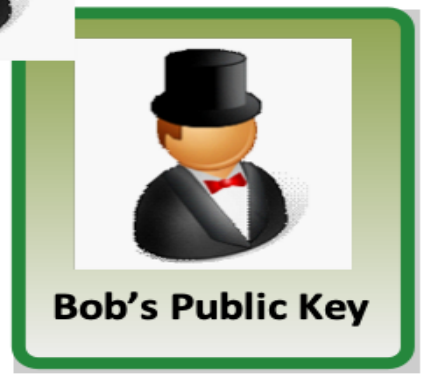


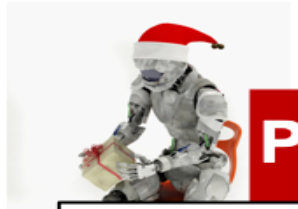


Levels of trust

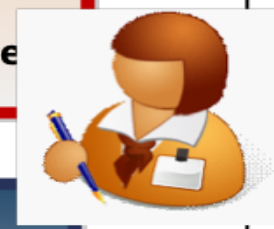
Public key encryption … secret … identity … trust

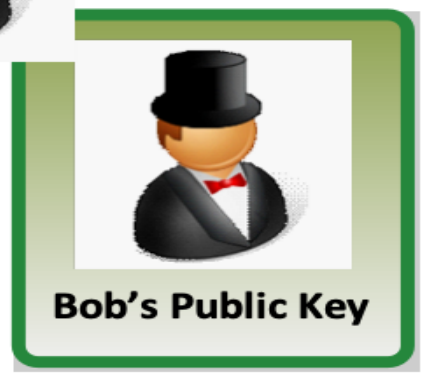




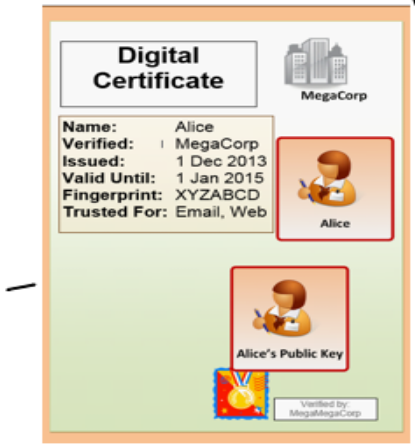


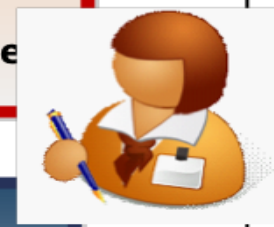
Public key encryption … secret … identity … trust

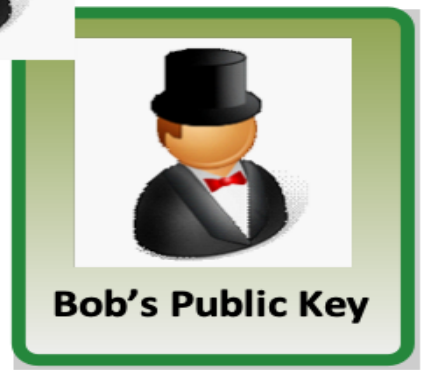


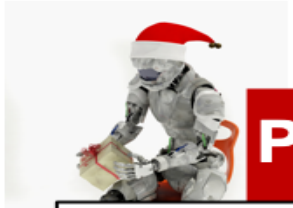


Public key encryption … secret … identity … trust





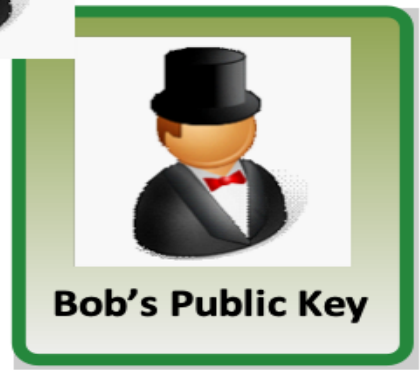


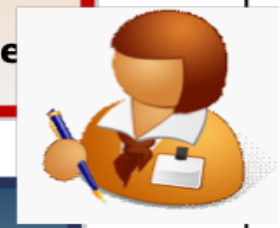


Public key encryption … secret … identity … trust

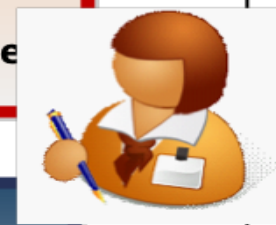
Hello Alice,
Wish you were here!
- Bob

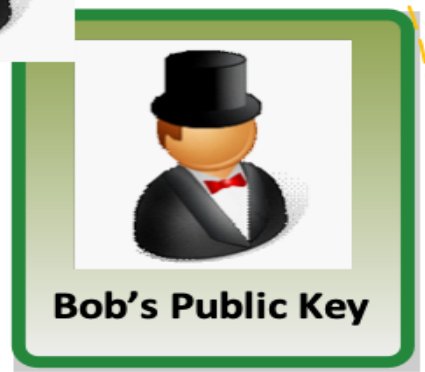
Bob.



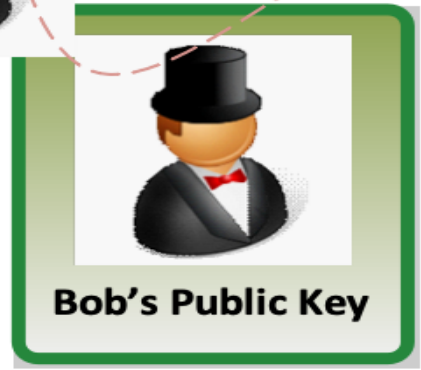


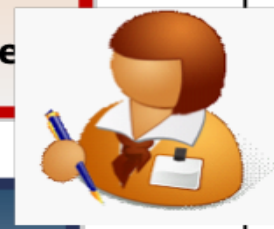
Public key encryption … secret … identity … trust

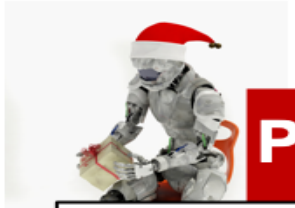




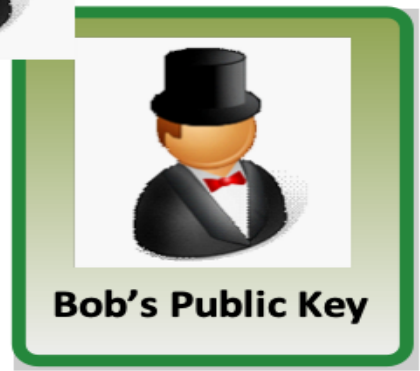
Public key encryption … secret … identity … trust



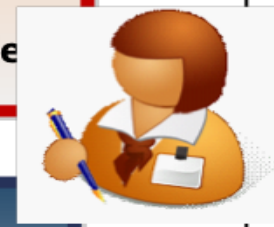


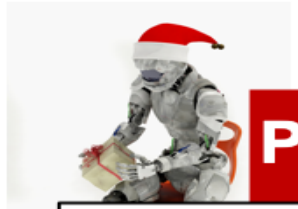


Public key encryption … secret … identity … trust



Which key to open the message?





Public key encryption … secret … identity … trust

Eve

Trent

MegaCorp

Digital Certificate

MegaCorp

Name: Alice
Verified: MegaCorp
Issued: 1 Dec 2013
Valid Until: 1 Jan 2015
Fingerprint: XYZABCD
Trusted For: Email, Web

Alice

Alice's Private Key

Alice's Public Key

Alice's Private Key

Alice's Public Ke

Bob's Private Key

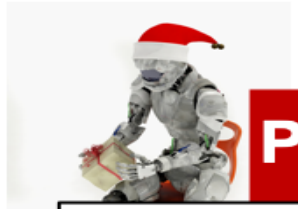
Hello Alice,
Wish you were here!
- Bob

Bob.

Which key to open the message?

Bob's Public Key

Alice's Private Key



Public key encryption … secret … identity … trust

Eve

Trent

MegaCorp

Bob's Private Key

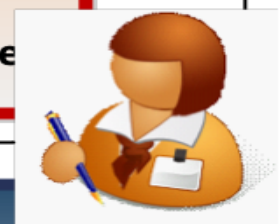
Bob's Public Key

Hello Alice,
Wish you were here!
- Bob

Bob.

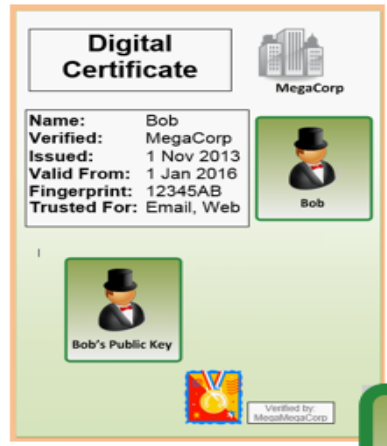
Which key to we open the signature with?

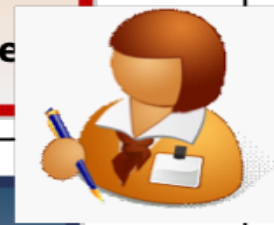
Alice's Public Ke

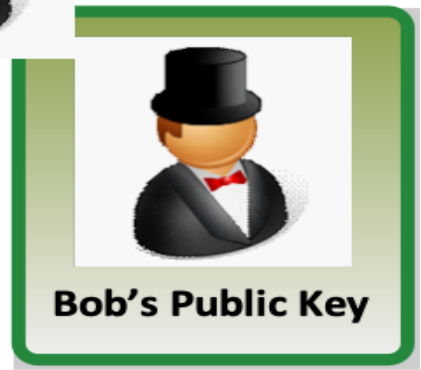


Alice's Private Key

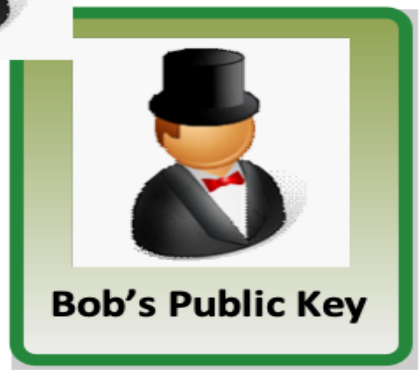
Public key encryption … secret … identity … trust

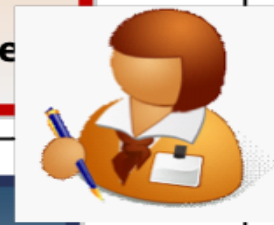


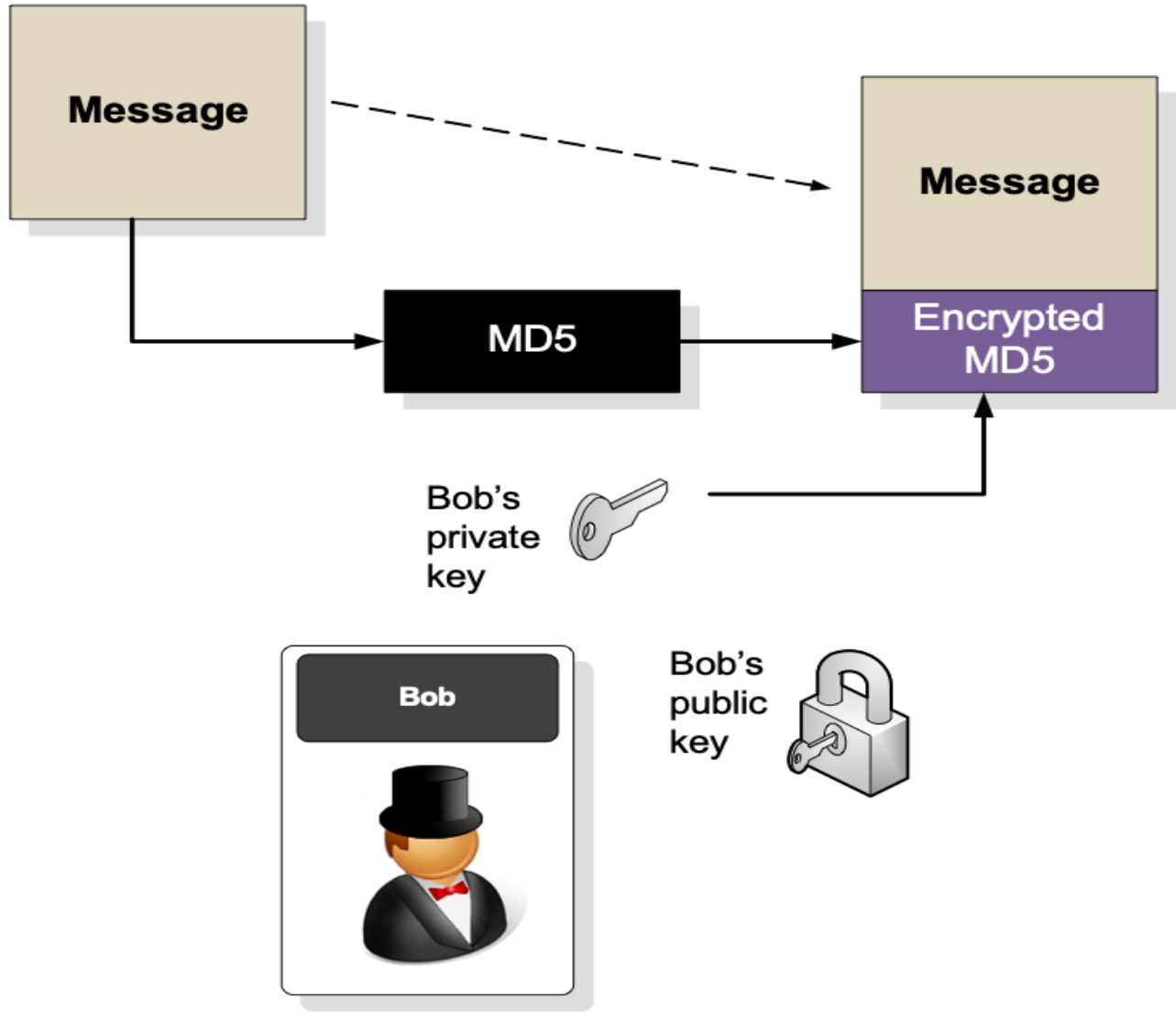




Public key encryption … secret … identity … trust

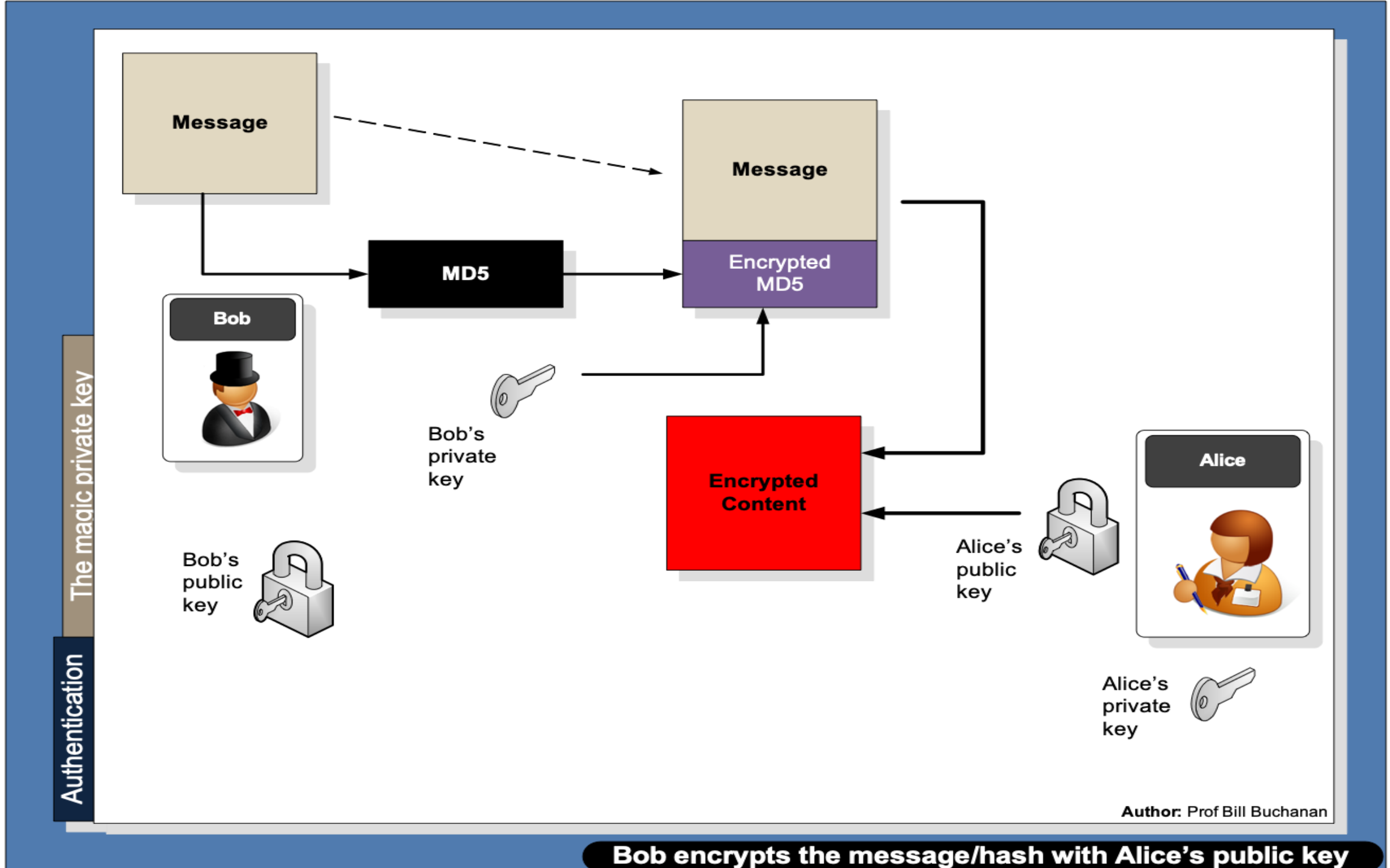


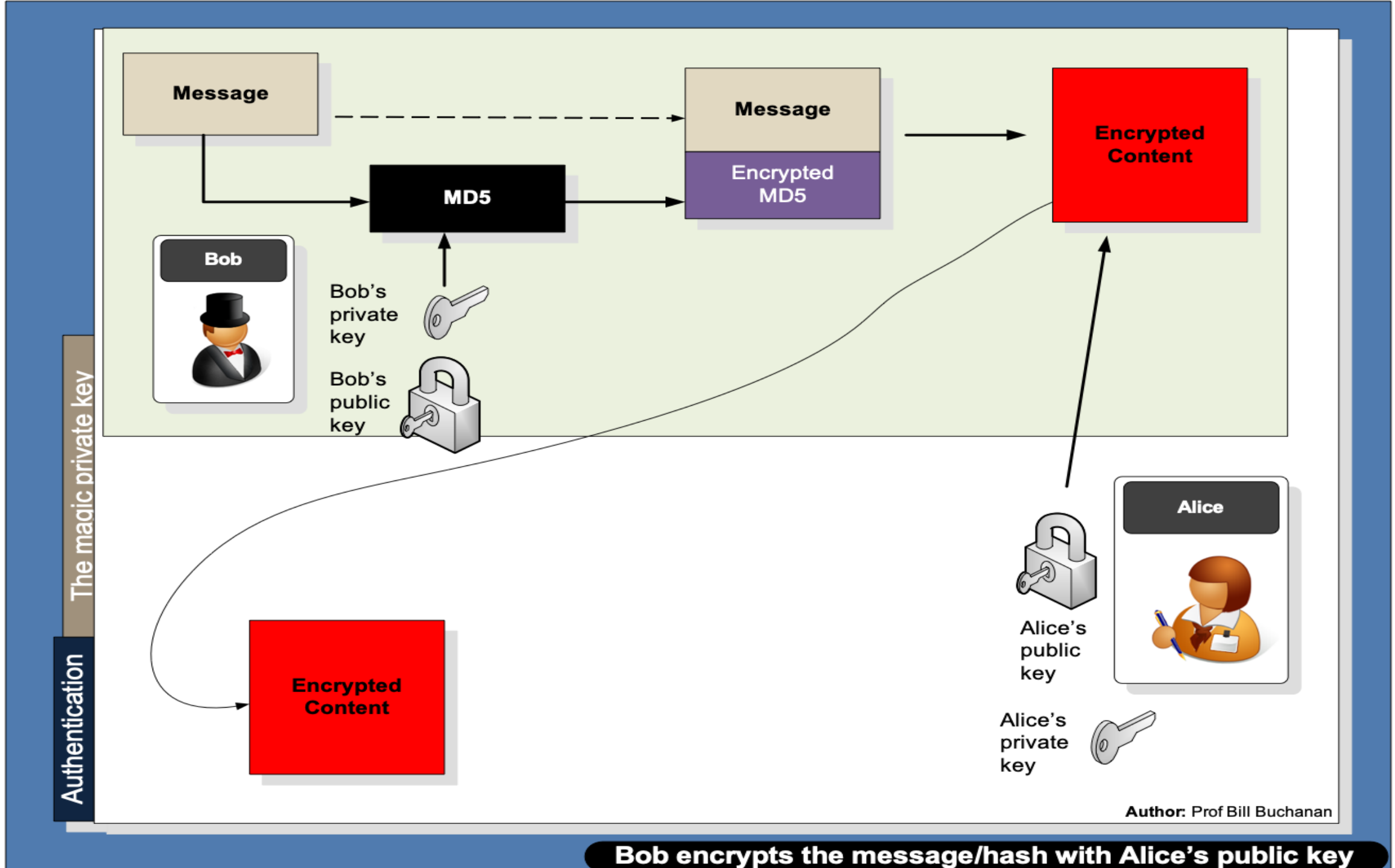


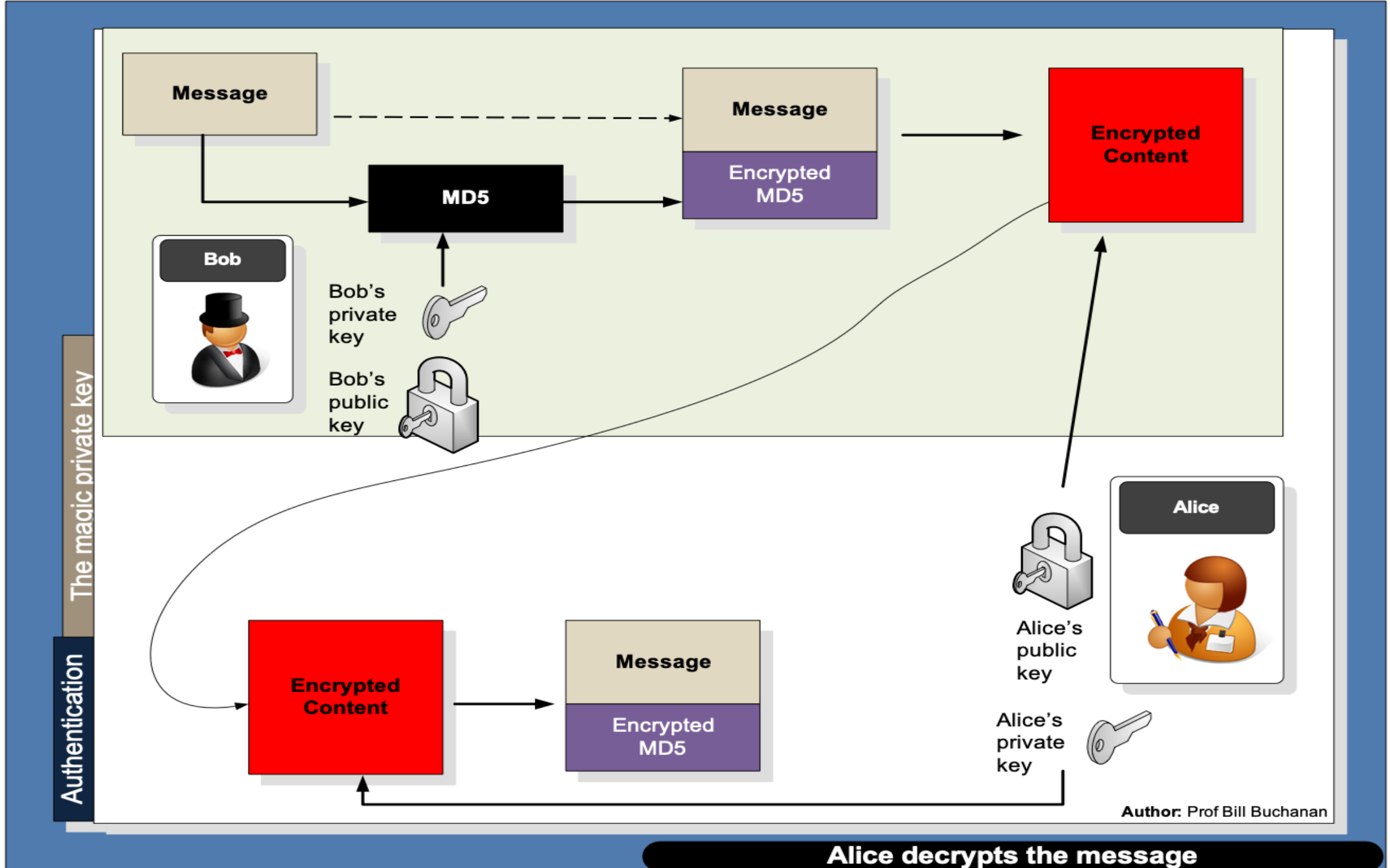


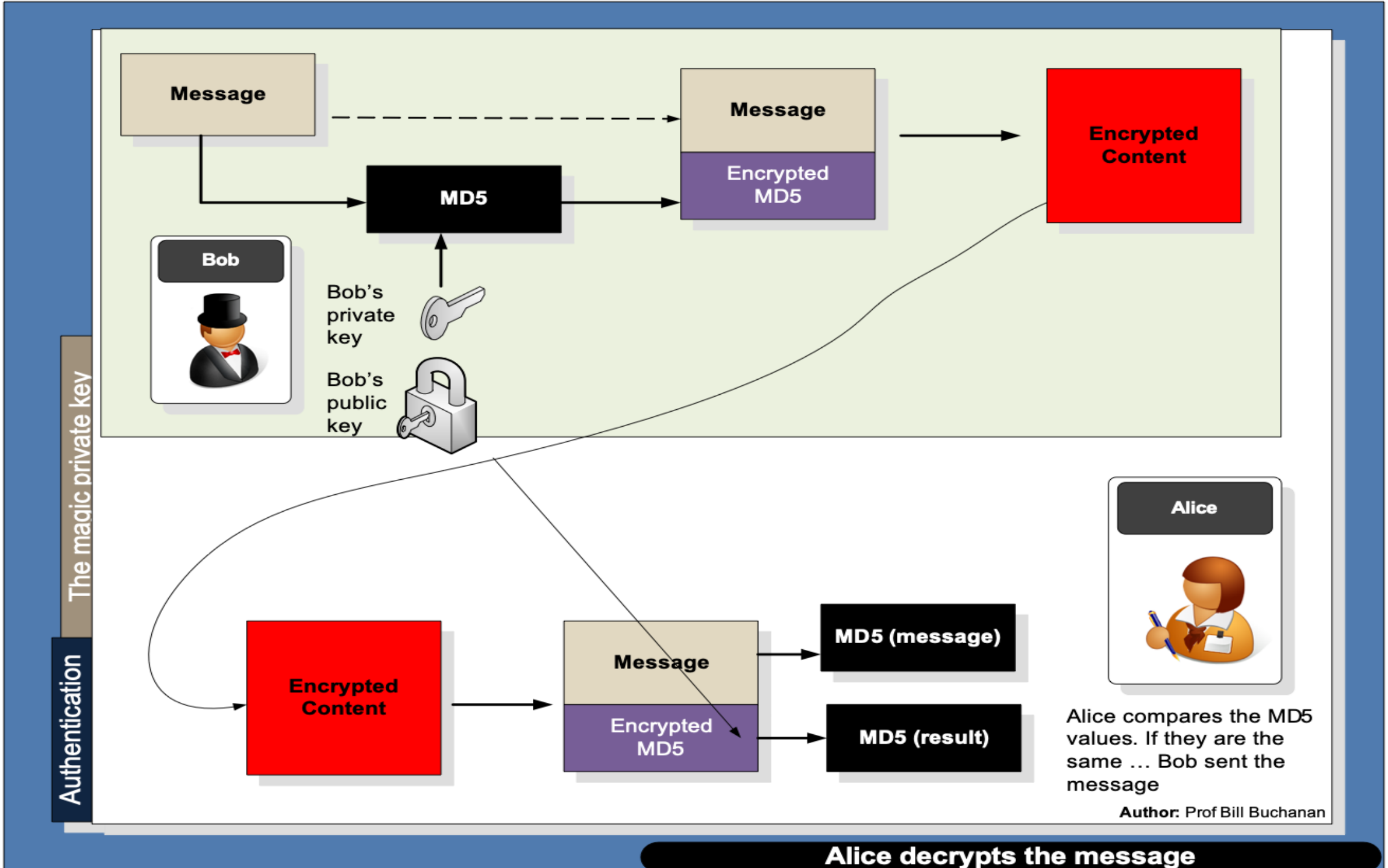
Author: Prof Bill Buchanan

Using Bob's private key to authenticate himself

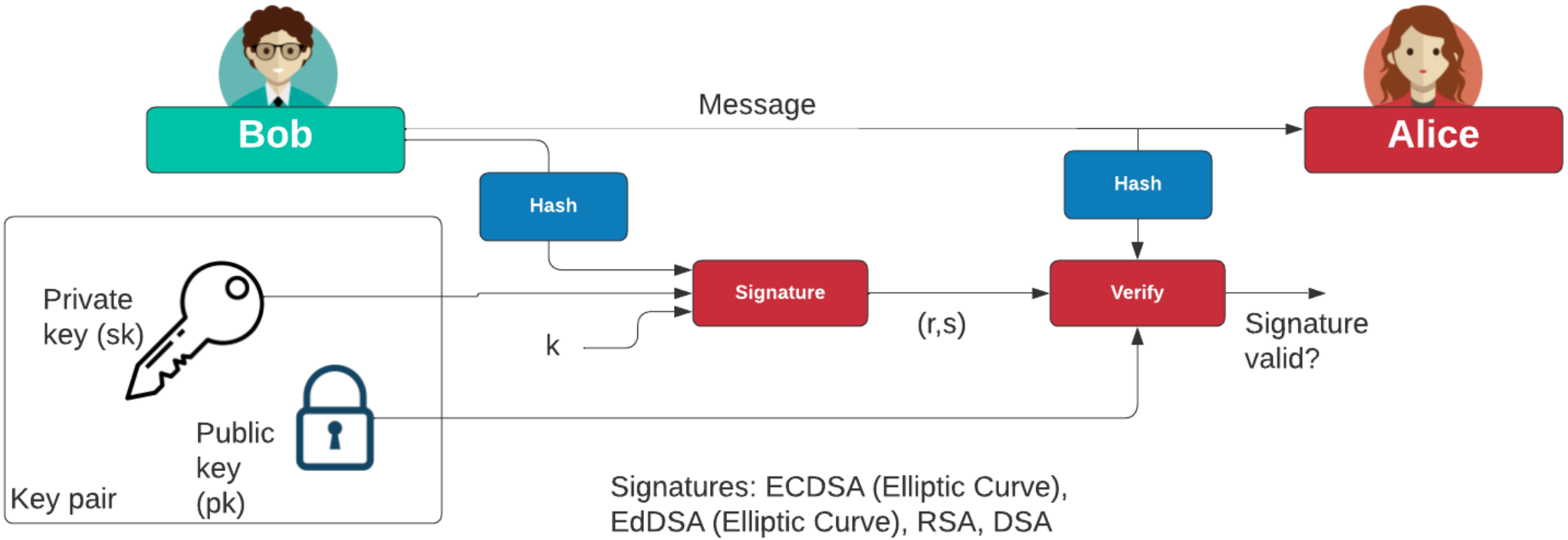

Message
Message
MD5
Encrypted MD5
Bob
Bob's private key
Encrypted Content
Alice
Bob's public key
Alice's public key
Alice's private key
The magic private key
Authentication
Author: Prof Bill Buchanan
Bob encrypts the message/hash with Alice's public key


Message
Message
Encrypted MD5
Encrypted Content
MD5
Bob
Bob's private key
Bob's public key
The magic private key
Authentication
Encrypted Content
Alice
Alice's public key
Alice's private key
Author: Prof Bill Buchanan
Bob encrypts the message/hash with Alice's public key

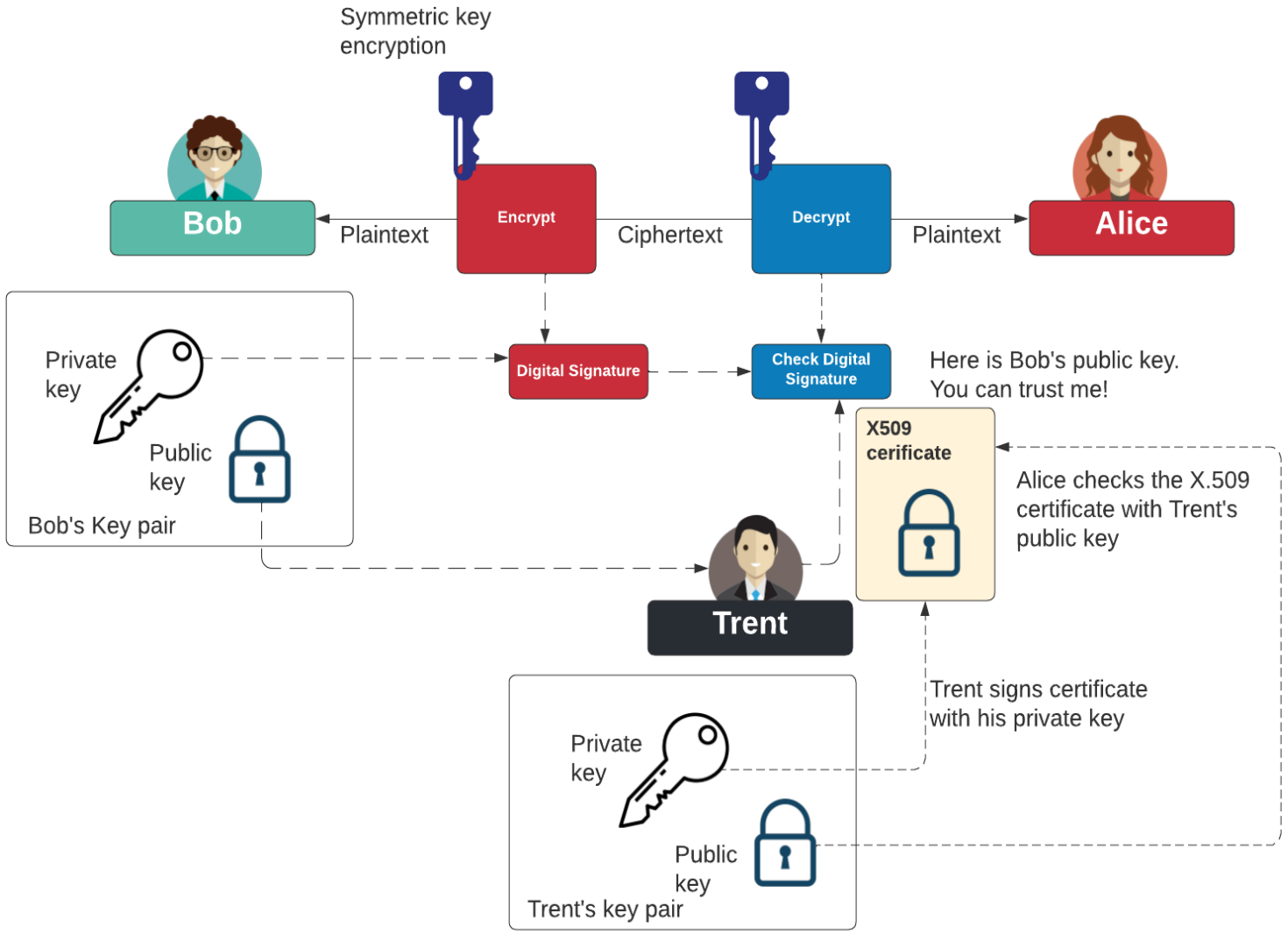

Message
Message
Encrypted MD5
Encrypted Content
MD5
Bob
Bob's private key
Bob's public key
The magic private key
Authentication
Alice
Alice's public key
Alice's private key
Encrypted Content
Message
Encrypted MD5
Author: Prof Bill Buchanan
Alice decrypts the message


Message
Message
Encrypted MD5
Encrypted Content
MD5
Bob
Bob's private key
Bob's public key
The magic private key
Authentication
Encrypted Content
Message
Encrypted MD5
MD5 (message)
MD5 (result)
Alice
Alice compares the MD5 values. If they are the same … Bob sent the message
Author: Prof Bill Buchanan
Alice decrypts the message

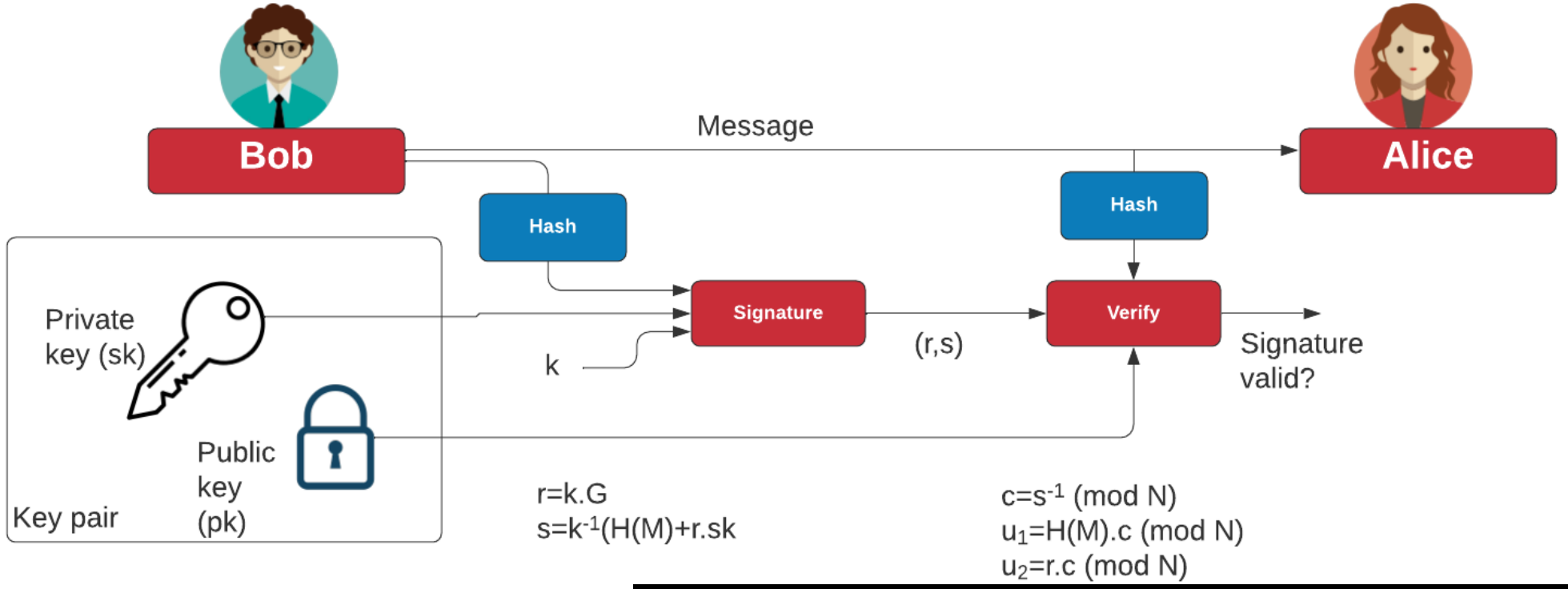
Overview


Bob
Message
Alice
Hash
Hash
Private
key (sk)
Signature
Verify
k
(r,s)
Signature
valid?
Public
key
(pk)
Key pair
Signatures: ECDSA (Elliptic Curve),
EdDSA (Elliptic Curve), RSA, DSA

Trent

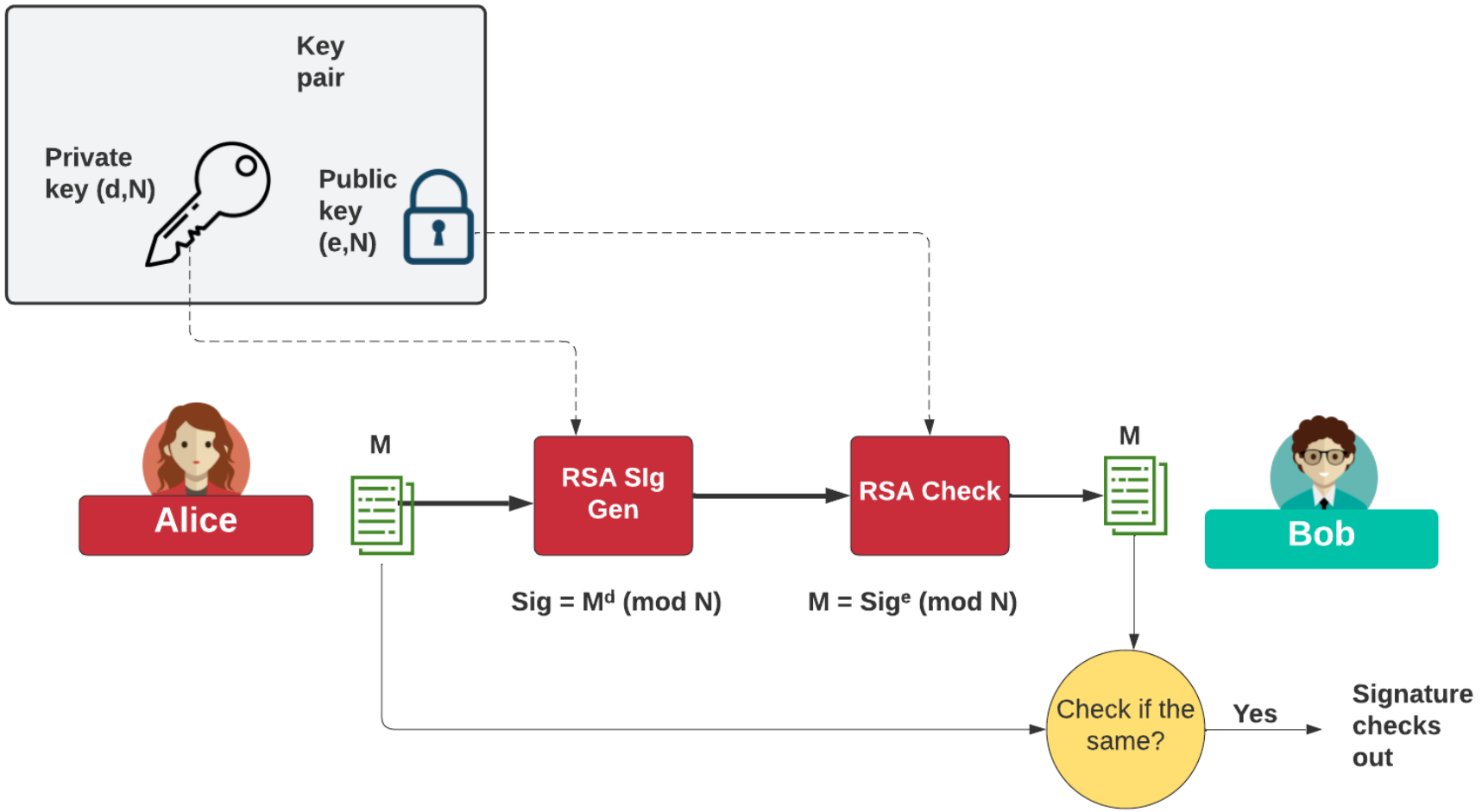

Symmetric key encryption
Bob
Plaintext
Encrypt
Ciphertext
Decrypt
Plaintext
Alice
Private key
Public key
Bob's Key pair
Digital Signature
Check Digital Signature
Here is Bob's public key. You can trust me!
X509 cerificate
Alice checks the X.509 certificate with Trent's public key
Trent
Trent signs certificate with his private key
Private key
Public key
Trent's key pair

ECDSA

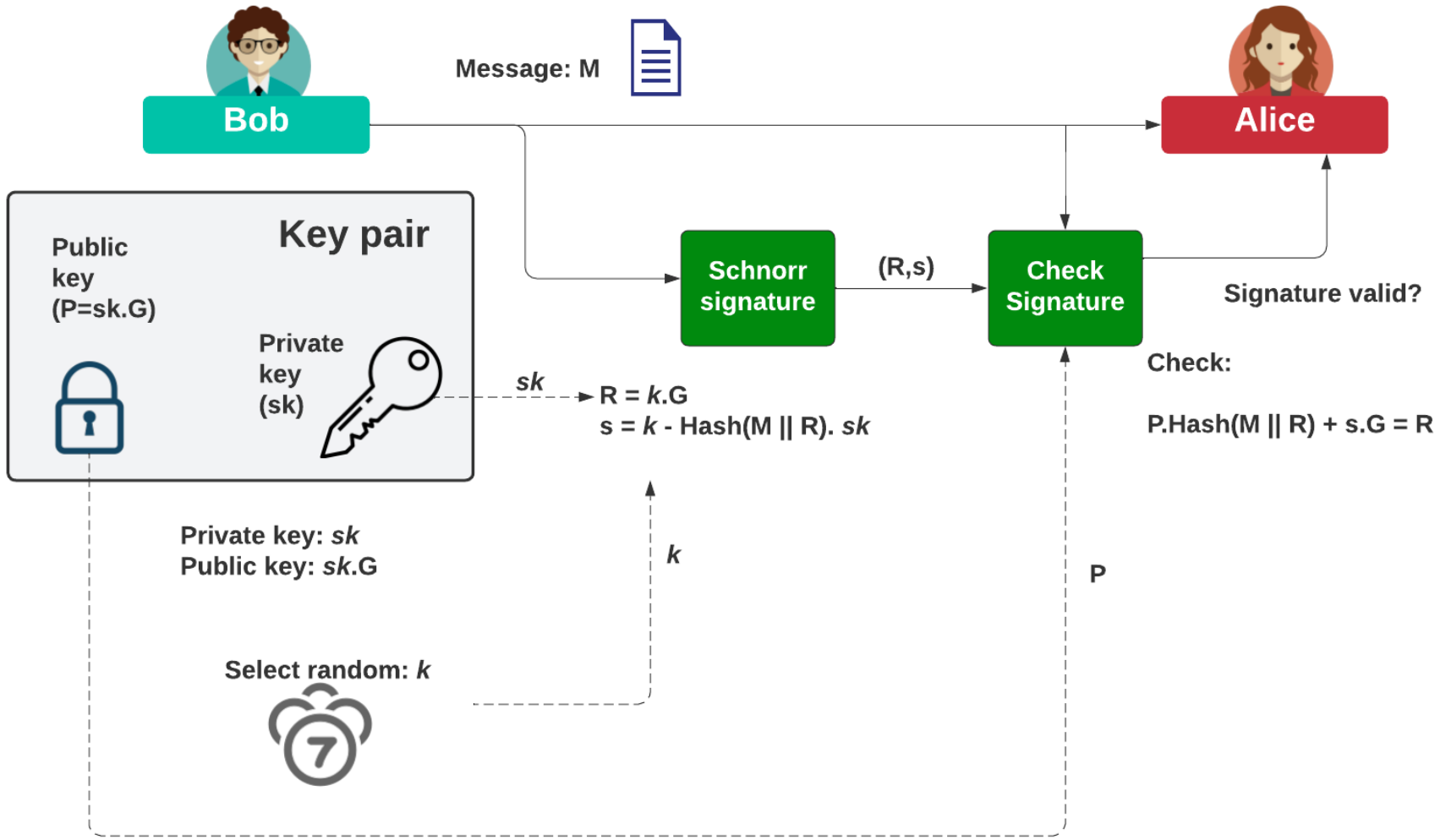



```
Msg: Hello

Alice's private key=914103191975445817821949313210686669640772316139623723952243372867878512076 23
Alice's public key=(107007219618086620558122760490508950019883614381385070624798268553636785085033,
50295619771883464285297058390425887963371353465625215351103778057382666752225)
k= 267011473021485390626792993924802270324838695218050221984055234315847884479 63

r=98607808147774449203575356285408019661538158474047478294689777633119075968396
s=99068437226471908177637999380627689653261729073056918331666171421799362411402

Result r=98607808147774449203575356285408019661538158474047478294689777633119075968396
Signature matches!
```

RSA Signatures



Schnorr Signatures – used with EdDSA



The Four Signatures

- ECDSA. Example.
- EdDSA (Ed25519). Example.
- RSA PKCS#1 v1.5. Example.
- RSA PSS (Probabilistic signature scheme). Example.

Cryptography

Prof Bill Buchanan OBE FRSE

https://asecuritysite.com

- Fundamentals
- Symmetric Key
- Hashing
- MAC
- KDF
- Public Key
- Key Exchange
- Signatures
- Digital Certificates

BLOCKPASS IDENTITY LAB

World-leading Collaboration between
Blockpass IDN and Edinburgh Napier University