Cryptographic Foundations of Cryptocurrencies

UAB COLLEGE OF ARTS AND SCIENCES
The University of Alabama at Birmingham

Department of
Computer Science

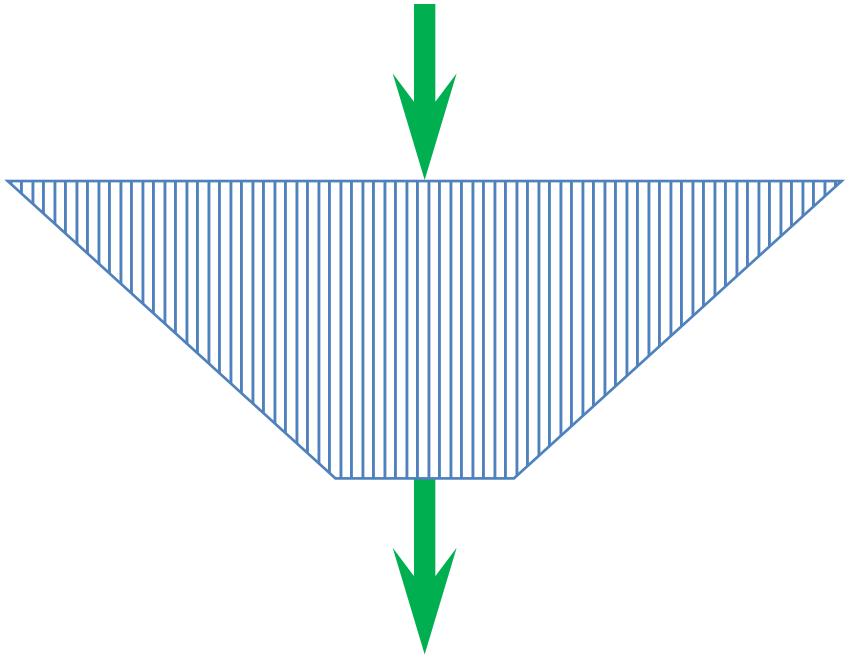
Basic Cryptographic Algorithms

- One-way hash algorithm
- Public key digital signature
- Elliptic curve cryptography

ONE-WAY HASH FUNCTION

One-Way Hash Algorithm

A document (of any length)



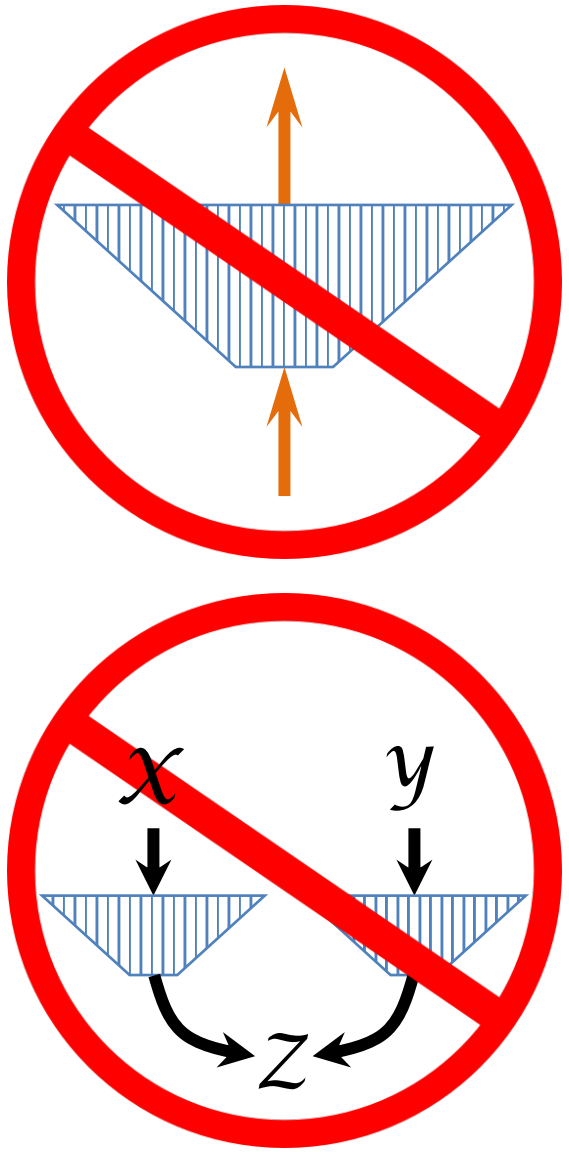
A condensed, short output
of fixed length (say of 256 bits)

One-Way Hash Algorithm

- It hashes an input document of any size into a condensed short output (say of 32 bytes)

- One-wayness
  - Given an output, it is infeasible for any one to find an input document which is hashed to that specific output !

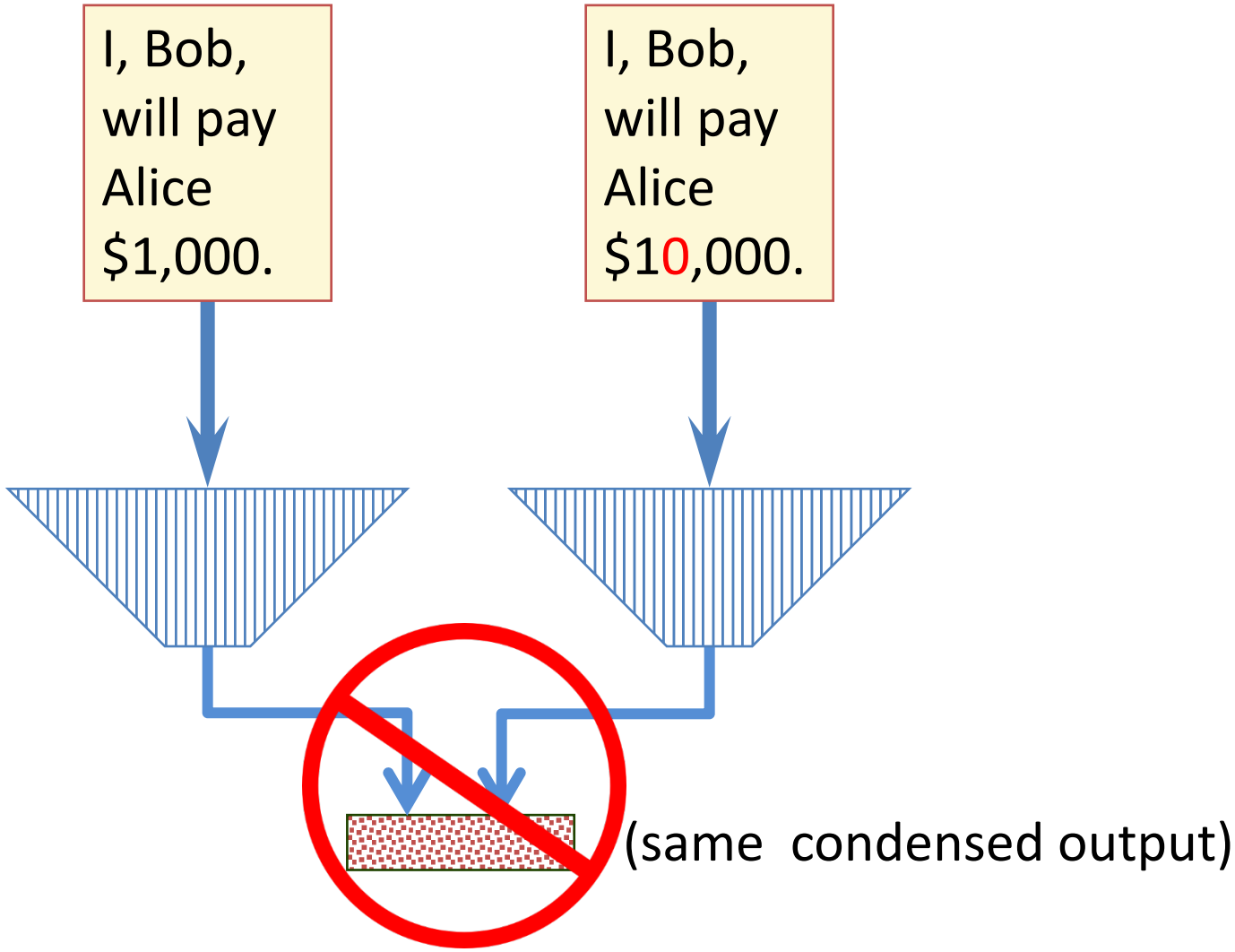
- Collision resistance
  - it is infeasible for any one to find two or more input documents which are hashed to the same condensed output !



2 Types of Collision

- Type 1
  - An attacker has one input document in hands
  - He attempts to find a different input document that is hashed to the same output as is the first input
- Type 2
  - An attack is not given any specific input document
  - His goal is find a pair of different input documents that are hashed to the same output
- A good 1-way hash should be immune to both types of attacks

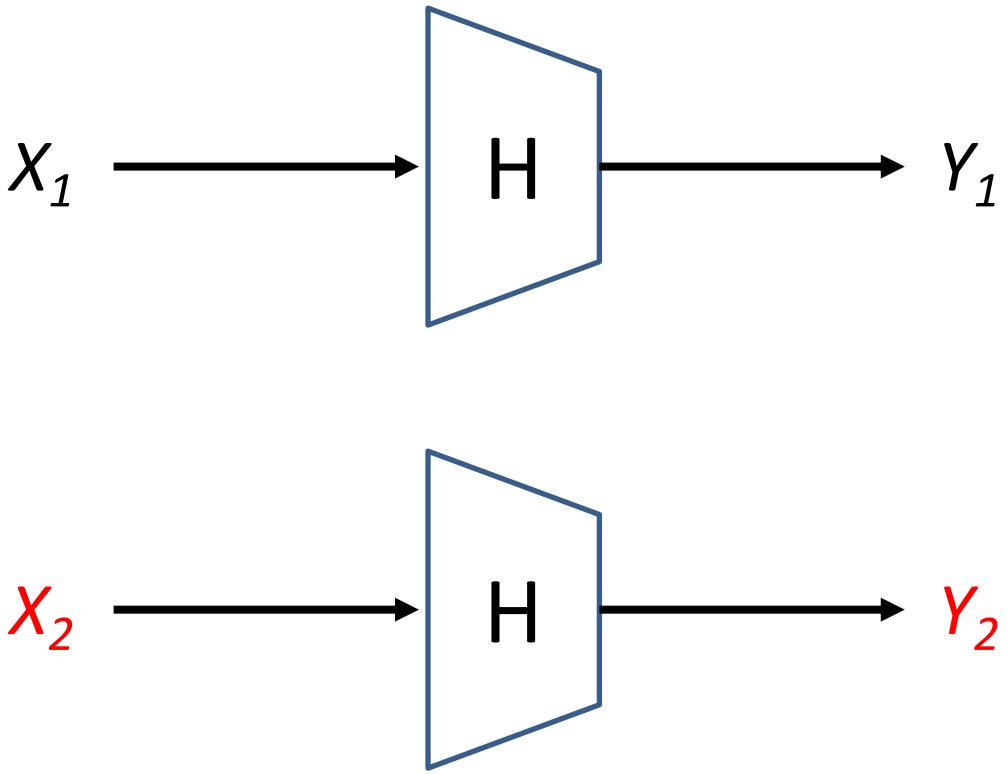
Finding Collision is Infeasible



Confetti Shredder/Wood Chipper as "1-Way Hash"

- Shredding a newspaper into very fine pieces
  - Pick & keep only 20 random pieces out of all those fine pieces & burn off the rest
- It's "1-way"
  - Infeasible for one to recover the original newspaper from the 20 fine pieces
- It's "collision-resistant"
  - Infeasible for one to find 2 different newspapers that are shredded to the same set of 20 pieces

1-Way Hash as a "Random Oracle"



1) Both Y_1 and Y_2 appear like a random value to all practical algorithms that do not "see" X_1 or X_2 .
That is Y_1 and Y_2 cannot be told apart from truly random ones.

2) Y_1 and Y_2 appear to be totally un-correlated, even when X_1 and X_2 are correlated.

Examples of one-way hash algorithms

Generation	Members	Year introduced	Still Secure?	Note
SHA1	SHA-160	1995	No	SHA0: 1993
SHA2	SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256	2001	Yes	
SHA3	SHA3-224, SHA3-256, SHA3-384, SHA3-512	2015	Yes	

Attacking 1-Way Hashing

Brute force attack

- **Taking advantage of "birthday paradox"**
- **This is a long route, or the *worst case* scenario from an attacker's point of view, but *the best case scenario* from the designer's point of view**

Attacks using short cuts

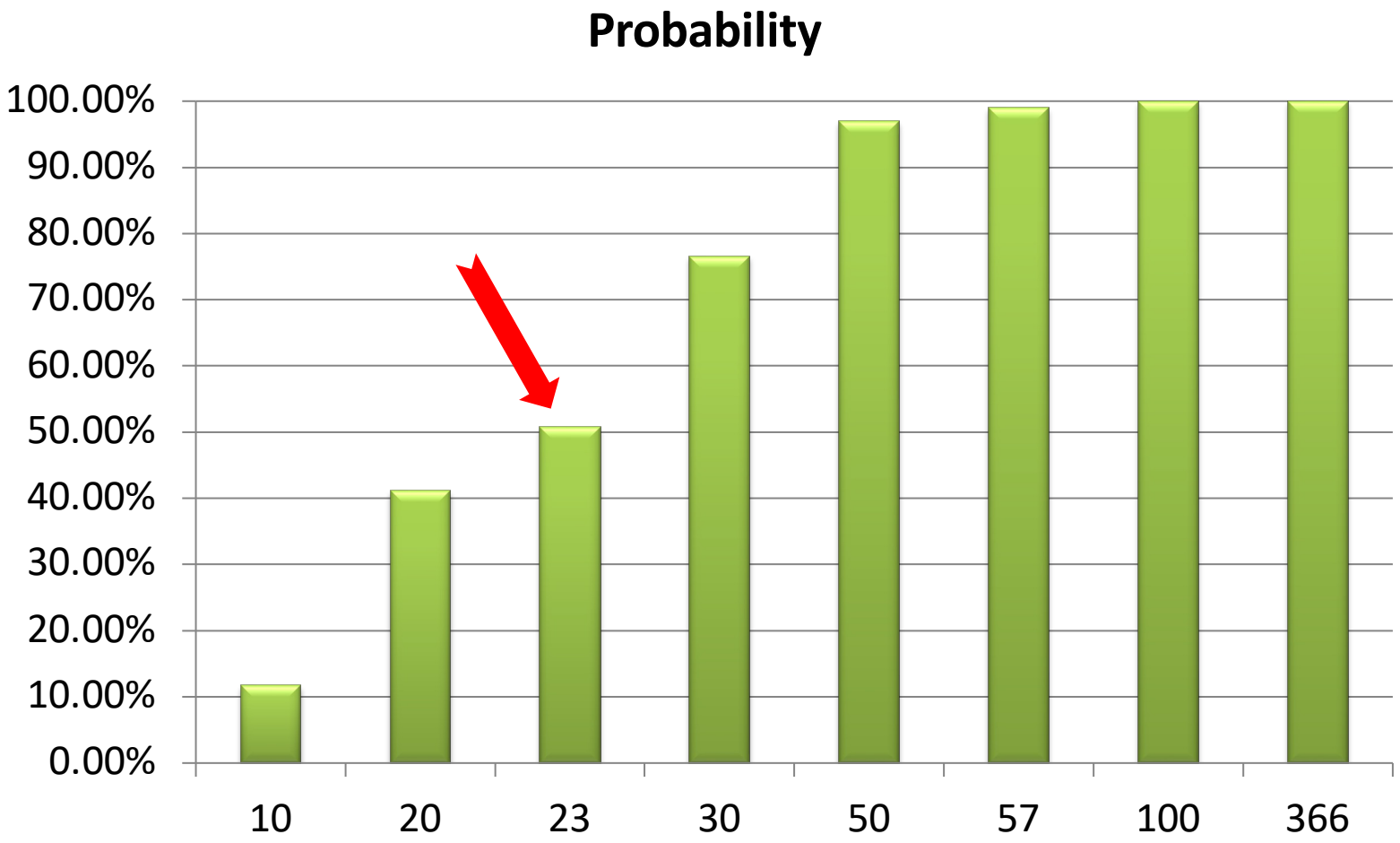
- **Identifying inherent weaknesses of a 1-way hashing**
- **The nightmare scenario from a desinger's point of view**

Birthday Paradox

- Given a bunch of messages selected at random, what's the chance that a pair of them might happen to collide (being hashed to the same output) ?

- Related to "Birthday problem":
  - Given a room of n people, what's the chance for a pair of them to be born on the same day?
  - Birthday paradox:
    - 23 people: Chance > 50%
    - 57 people: Chance > 99%

Birthday Paradox



n	*P*(*n*)
5	2.70%
10	11.70%
20	41.10%
23	50.70%
30	70.60%
40	89.10%
50	97.00%
60	99.40%
70	99.90%
366	100%

Birthday Attack: the Long Route to Rome

- Basic idea
  - Compute hashes of many messages,
  - Store digest-message pairs in memory (digest=hash tag or msg fingerprint),
  - Use a fast sorting algorithm to rearrange them in the order of digests
- Successful if there are identical digests in the sorted list
- For 1-way hashing with *t*-bit digest, the expected # of messages to be hashed before a coincidence/collision occurs is

$$1.25 \bullet 2^{t/2}$$

- E.g #1.
  - $t = 64$, it takes approx. 2^{32} computational steps to find a successful collision
- E.g #2.
  - $t = 256$, it takes approx. 2^{128} computational steps to find a successful collision
- Note: the attacker can do better using "time-memory trade-off"

Exploiting Inherent Weaknesses: the Short Cut to Rome

- Requires *deep insights* into 1-way hashing, mathematics, cryptanalysis, computation.
- Weaknesses were found in MD5 in August 2004
  - Numerous collisions were produced
- Weaknesses were found with SHA-1 (SHA-160) in February 2005,
  - collisions could be found in $< 2^{69}$ steps, rather than 2^{80} steps required by brute-force/birthday attack !
  - A reduced round version of SHA-1 (58 rounds) could be broken in $< 2^{33}$ steps.

Comparison of SHA Siblings

One-Way Hash Algorithm	Message size (bits)	Block size (bits)	Word size (bits)	Hash output size (bits)	# of rounds	Brute force (birthday) attack	Attacks exploiting inherent weakness
SHA-1 (SHA-160)	$< 2^{64}$	512	32	160	**80**	2^{80}	$< 2^{69}$
SHA-224 (truncated SHA-256)	$< 2^{64}$	512	32	224	**64**	2^{112}	?
SHA-256	$< 2^{64}$	512	32	256	**64**	2^{128}	?
SHA-384 (truncated SHA-512)	$< 2^{128}$	1024	64	384	**80**	2^{192}	?
SHA-512	$< 2^{128}$	1024	64	512	**80**	2^{256}	?

References for 1-Way Hash

1. R. Rivest, The MD5 Message-Digest Algorithm, RFC 1321, 1992, available at www.ietf.org/rfc/
2. FIPS PUB 180-4, *Specification for Secure Hash Standard,* 2015, available at https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

RSA Public Key Encryption

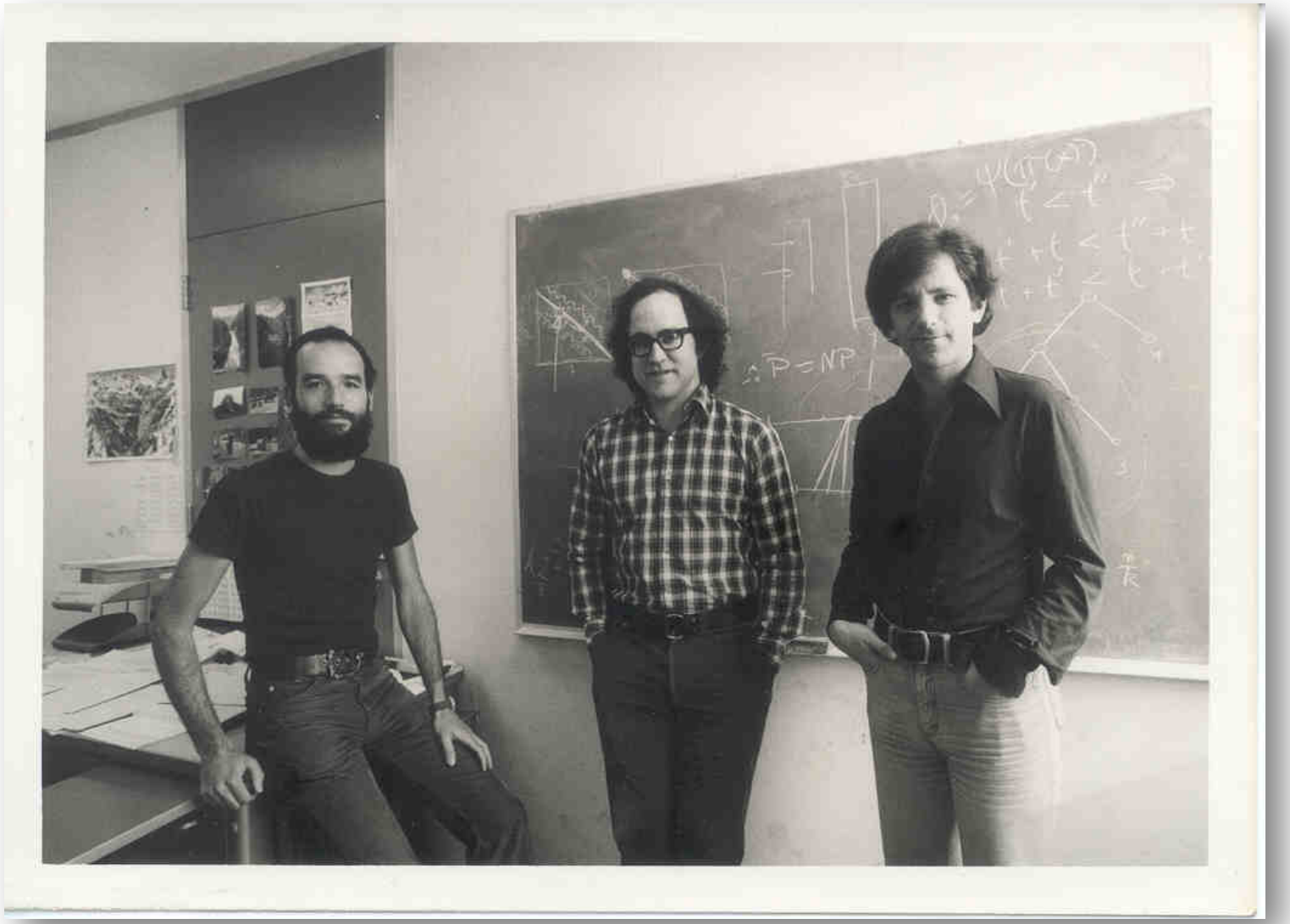
CACM, February 1978, Vol. 21 No. 2

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

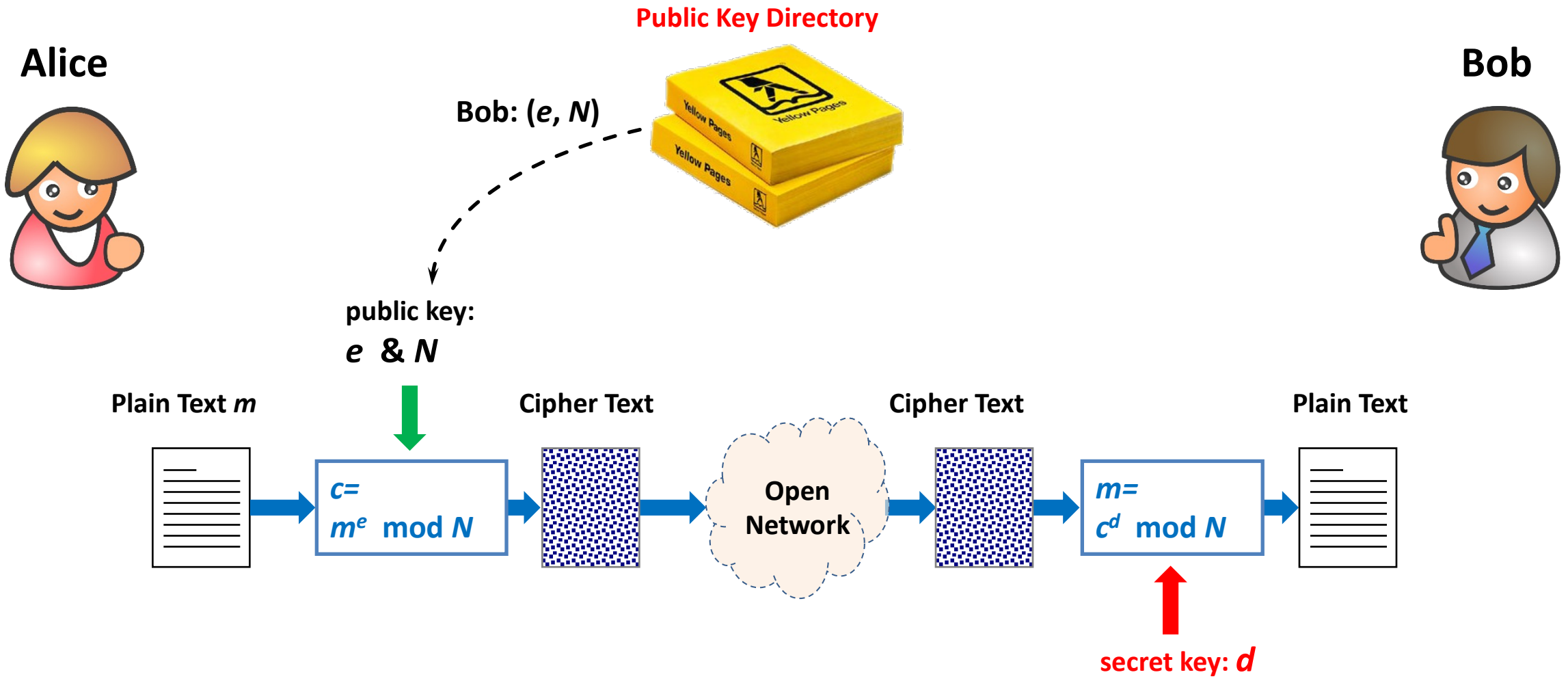
An encryption method is presented with the novel property that publicly revealing an encryption key does not thereby reveal the corresponding decryption key. This has two important consequences:
(1) Couriers or other secure means are not needed to transmit keys, since a message can be enciphered using an encryption key publicly revealed by the intended recipient. Only he can decipher the message, since only he knows the corresponding decryption key.

S R A = $(RSA)^e$ mod *N*



Adi Shamir Ron Rivest Leonard Adelman

RSA Public Key Cryptosystem

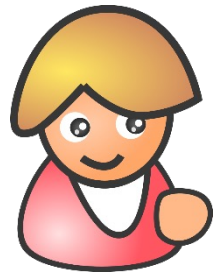


RSA Public key encryption

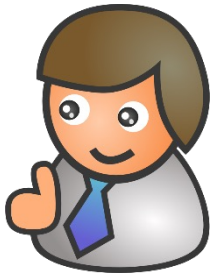
Preparation by Bob

- Choose 2 large primes (each at least 300 digits): p, q
- Multiply p and q: $N = p*q$
- Find out two numbers e and d such that
  $e * d = 1 \pmod{\varphi(N)}$
  where $\varphi(N) = (p-1)(q-1)$
- Publish (e, N) as his public encryption key
- Save d as his matching secret decryption key (together with N)

Encryption & Decryption



- Alice wants to send a message m to Bob:
  - Find out Bob's public encryption key (e, N)
  - Compute
    $c = m^e \pmod{N}$
  - Send the ciphertext c to Bob
- Bob receives c from Alice
  - Use his secret decryption key d to calculate
    $m = c^d \pmod{N}$



RSA --- a Toy Example

- **Bob prepares the following:**
  - Choose 2 primes: $p=5, q=11$ multiply p and q:
    $N = p*q = 55$
  - Find out two numbers $e=3$ & $d=27$ which satisfy
    $3 * 27 = 1 \pmod{40}$
  - Bob's public key
    - 2 numbers: (*3, 55*)
  - Bobs' secret key:
    *27 (*and *55)*

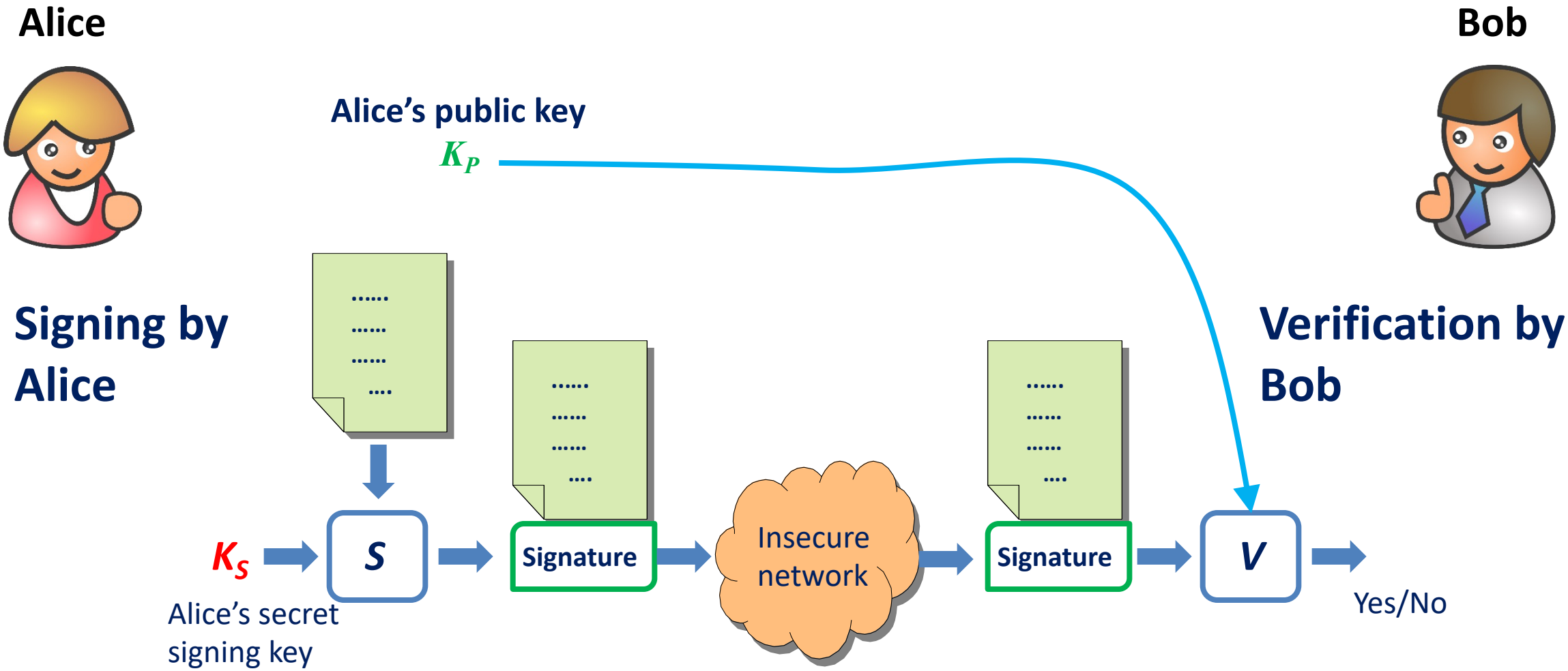
- **Alice wants to send *m=13* to Bob:**
  - Fetch Bob's public encryption key (*3, 55*)
  - Encrypt

    $c = m^e \pmod{n}$
    $= 13^3 \pmod{55}$
    $= 2197 \pmod{55}$
    $= 52$
  - Send *52* to Bob
- **Bob receives *52* from Alice**
  - Use his secret decryption key *27* to recover

    $m = 52^{27} \pmod{55}$
    $= 13$

Public Key Digital Signature

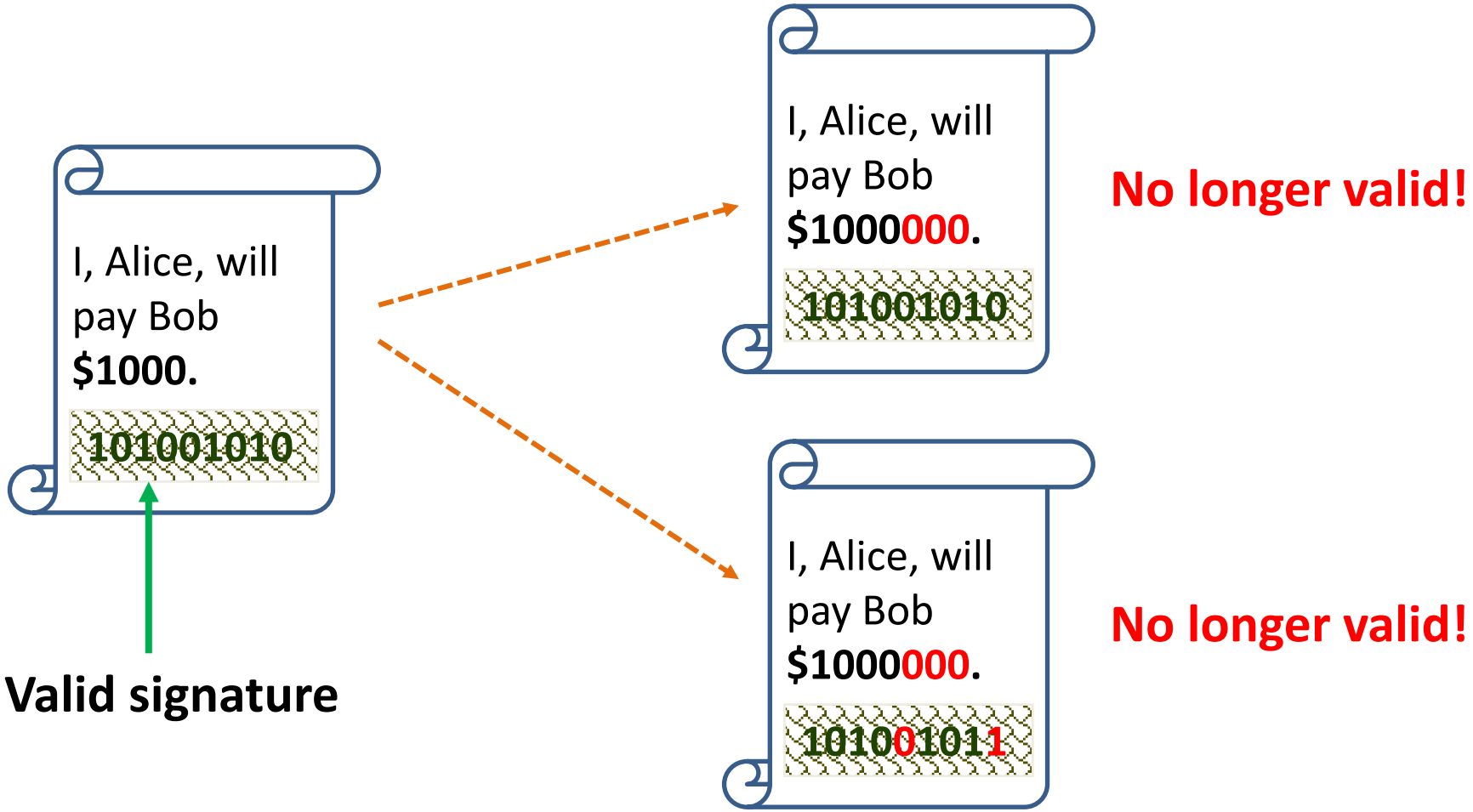
Public key digital signature



Useful Properties of Digital Signature

- Unforgeable
  - takes 1 billion years to forge !
- Un-deniable by the signatory
- Universally verifiable
- Differs from document to document
- Easily implementable by
  - software or
  - hardware or
  - software + hardware

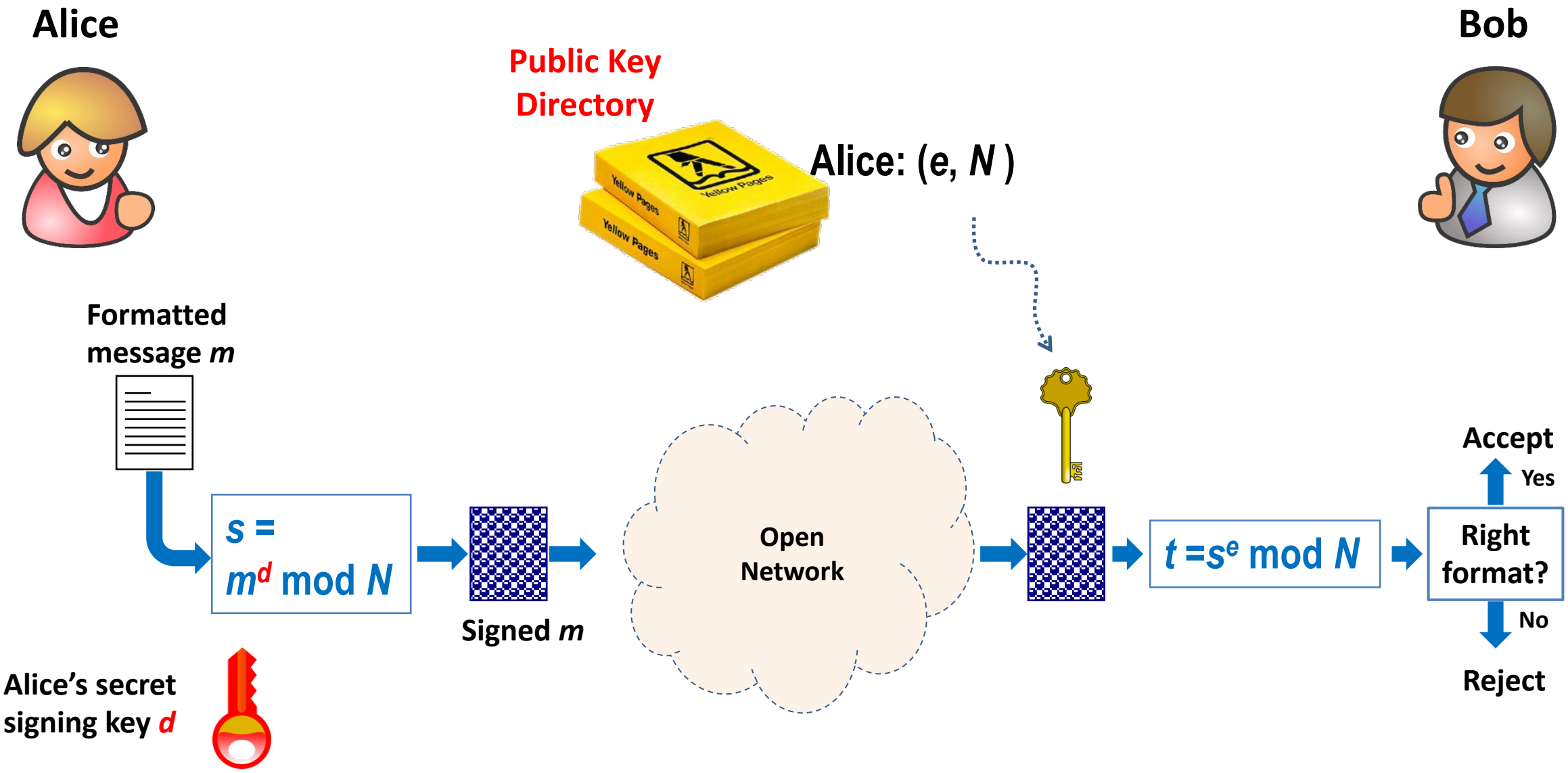
Unforgeable Digital Signature



Important Digital Signatures

- RSA
- DSS (digital signature standard, USA)
  - derived from ElGamal digital signature
  - based on infeasibility of discrete logarithm
  - strongly pushed forward by US government
- Schnorr digital signature
  - derived from ElGamal digital signature
  - based on infeasibility of discrete logarithm
- Signature schemes using elliptic curves

RSA Digital Signature with message recovery

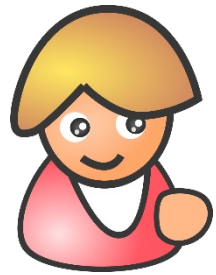


RSA Digital Signature

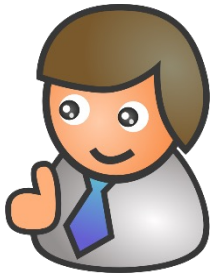
Preparation by Alice

- Choose 2 large primes (each at least 300 digits): p, q
- Multiply p and q: $N = p*q$
- Find out two numbers e and d such that
  $e * d = 1 \pmod{\varphi(N)}$
  where $\varphi(N) = (p-1)(q-1)$
- Publish (e, N) as his public verification key
- Save d as his matching secret signing key (together with N)

Signing & Verification



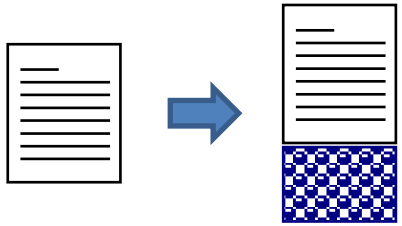
- Alice signs a (formatted) message m
  - Compute
    $s = m^d \pmod{N}$
  - Send the signed msg s to Bob
- Bob receives s from Alice
  - Find out Alice's public verification key (e, N)
  - Calculate
    $m = s^e \pmod{n}$
    and accept it iff m is in right format



RSA Signature in practice

- **RSA-PSS-R**
  - With message recovery (no appendix)
  - Good for short messages only
  - Use salt to randomize signature
  - Efficient, and good for digital cash

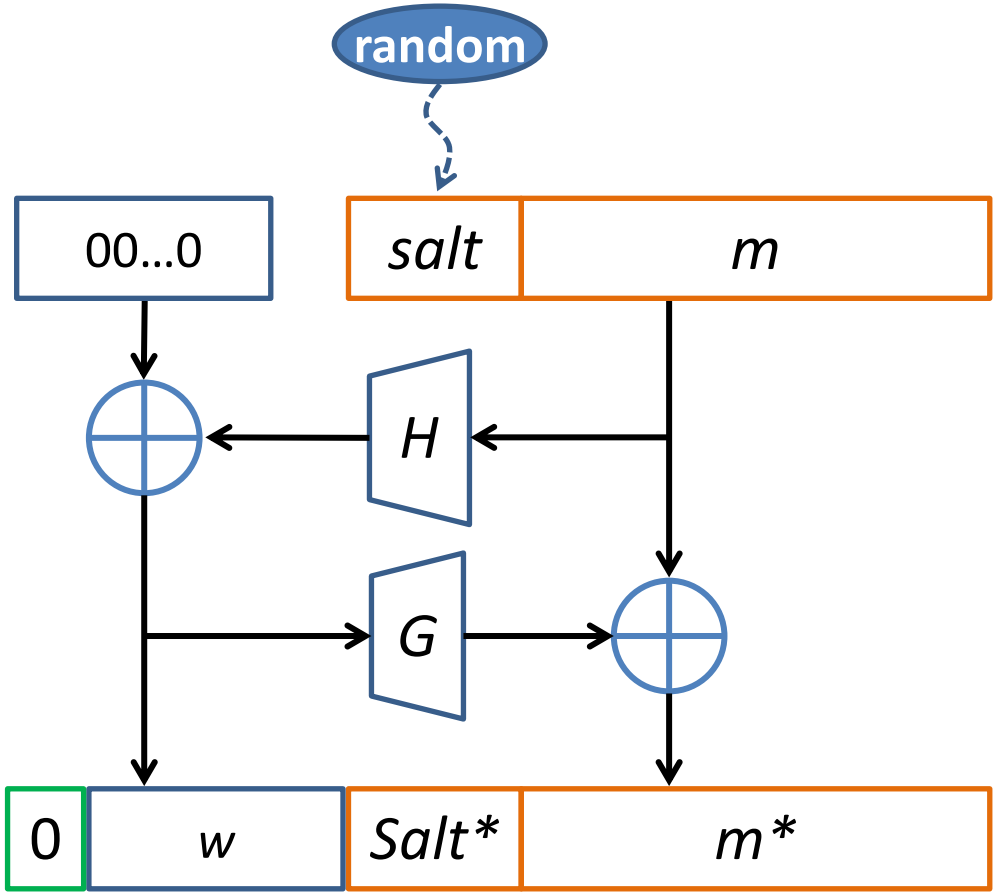
- **RSA-PSS**
  - Sign a message with an appendix
  - Good for messages of arbitrary length



- **Ref:**
  M. Bellare and P. Rogaway. The Exact Security of Digital Signatures - How to Sign with RSA and Rabin. In Advances in Cryptology - Eurocrypt '96, LNCS, Vol. 1070, pp. 399 - 416. Springer Verlag, 1996.

RSA-PSS-R: formatting a message

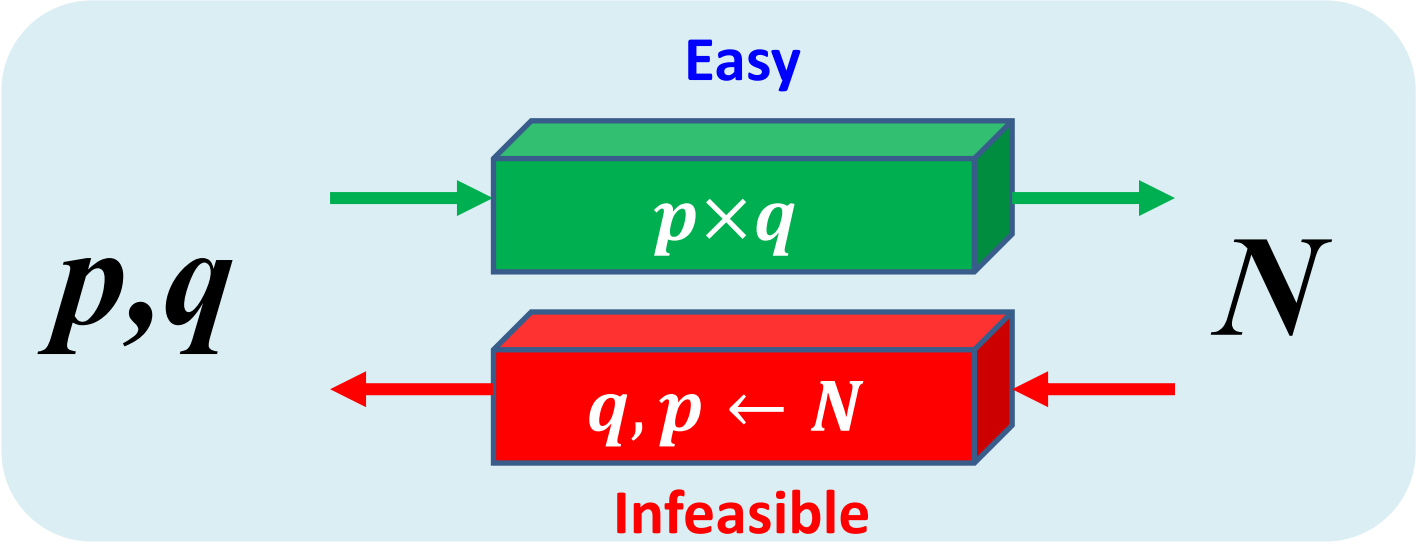
- **Signing by Alice**
  - Transform a message m into $t = 0 \parallel w \parallel salt^* \parallel m^*$
  - $s = t^d \bmod N$
- **Verifying by Bob**
  - Break up s into $(b, w, salt^*, m^*)$
  - $(salt, m) = G(w) \oplus (salt^* \parallel m^*)$
  - Accept iff $b = 0$ and $w = H(salt \parallel m)$



Why is RSA signature considered secure

- **RSA is based on the factorization problem:**

  While it is easy to multiply large primes together, it is computationally infeasible to factorize or split a large composite into its prime factors !



Parameter sizes for RSA

Security Level (in bits)	80	112	128	192	256
p	512	1024	1536	3840	7680
q	512	1024	1536	3840	7680
$N=pq$	1024	2048	3072	7680	15360

Digital Signature Standard (DSS)

- **Based on Discrete Logarithm**
- **A US standard:**
  - ***FIPS 186, 19/5/1994*.**
  - ***FIPS 186-2, 11/2000*.**
  - ***FIPS 186-3, 6/2009*.**
- **Like Schnorr signature, DSS is a modified version of ElGamal signature**

DSS - Alice's Public Key (similar to Schnorr)

- **Alice's public key = (y, p, q, g) where**
  - **p = a prime of at least 2048 bits.**
  - **q = a 160-bit prime divisor of $p - 1$.**
  - **$g = h^{(p-1)/q} \bmod p$, where h is any integer with $1 < h < p - 1$ s.t. $h^{(p-1)/q} \bmod p > 1$. That is g has order $q \; mod \; p$.**
  - **$y = g^x \bmod p$, where x is an integer randomly selected from $[1, q - 1]$.**

DSS - Alice's Private Key

- **Alice's private key which is used in signature generation is the integer x selected at raondom from $[1, q-1]$.**
- **Note the relationship between Alice's private and public keys:**

$$y = g^x \bmod p,$$
$$0 < x < q.$$

DSS - Signing a document m by Alice

- To sign a document m, Alice does the following:
  - Pick at random k from $[1, q-1]$.
  - $r = (g^k \bmod p) \bmod q$
  - $s = (k^{-1} * (HASH(m) + x * r)) \bmod q$, where HASH is a 1-way hash whose output is 160 bits.
- Alice's signature on m is the pair of numbers (r, s).

DSS - Verification by Bob

- To verify (m, r, s), Bob does the following:
  - Fetch Alice's public key (y, p, q, g) from the public key file
  - $w = s^{-1} \bmod q$
  - $a = HASH(m) * w \bmod q$
  - $b = r * w \bmod q$
  - $v = (g^a * y^b \bmod p) \bmod q$
  - OK only if $v = r$.

DSS - Summary

- Alice's private signing key = x chosen at random from $[1, q-1]$
- Alice's public key = (y, p, q, g), where $y = g^x \bmod p$

- Signature on m
  - $r = (g^k \bmod p) \bmod q$
  - $s = (k^{-1} * (HASH(m) + x * r)) \bmod q$ where k is selected at random from $[1, q-1]$.

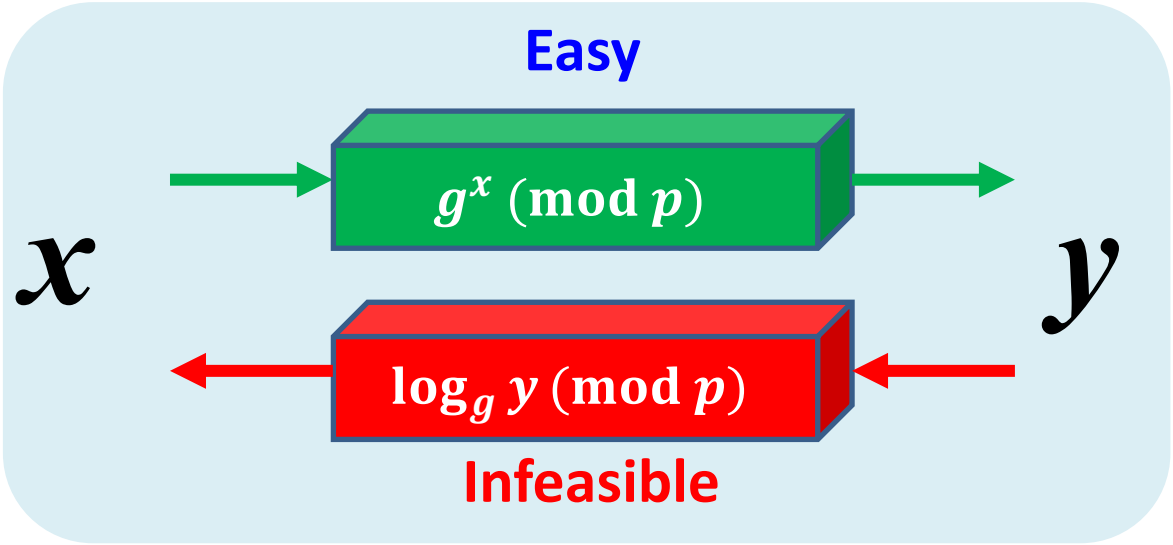
- Verification of (m, r, s)
  - check whether r is identical to $(g^a * y^b \bmod p) \bmod q$

  where
  - $a = HASH(m) * w \bmod q$
  - $b = r * w \bmod q$
  - $w = s^{-1} \bmod q$

Security of Digital Signature Standard (DSS)

- **Security of DSS rests on the difficulty of computing Discrete-Logarithm in a large finite field.**

- **When p is large (say ~700 digits), the best known algorithm for the Discrete-Logarithm Problem takes an infeasible amount of time (the problem is computationally infeasible).**



Parameter sizes for DSS

Security Level (in bits)	80	112	128	192	256
p	1024	2048	3072	7680	15360
q	160	224	256	384	512
Hash Output	160	224	256	384	512

Schnorr signature & Schnorr Multi-signature

Schnorr Signature – system wide parameters

- System wide: (p, q, g)
  where
  - p = a prime of at least 2048 bits.
  - q = a 160-bit prime divisor of $p - 1$.
  - $g = h^{(p-1)/q} \bmod p$,
    where h is any integer with $1 < h < p - 1$ s.t.
    $h^{(p-1)/q} \bmod p > 1$.
    That is g has order $q \bmod p$.

Schnorr signature: (c, s) version

- Parameters
  - p = a prime of at least 2048 bits.
  - q = a 160-bit prime divisor of $p - 1$.
  - $g = h^{(p-1)/q} \bmod p$, where h is any integer with $1 < h < p - 1$ s.t. $h^{(p-1)/q} \bmod p > 1$. That is g has order $q \bmod p$.
  - *Hash* is a 1-way hash function
- Alice's private key = x, chosen at random from $[1, q - 1]$
- Alice's public key = (y, p, q, g), where $y = g^x \bmod p$
- Signature on m is a pair (c, s)
  - $r = g^k \bmod p$, where k is chosen at random from $[1, q - 1]$.
  - $c = Hash(r, m)$.
  - $s = (k + c * x) \bmod q$.
- Verification of (m, c, s)
  - Check if c is identical to $Hash((g^s * y^{-c} \bmod p), m)$

Schnorr signature: (r, s) version

- **Parameters**
  - p = a prime of at least 2048 bits.
  - q = a 160-bit prime divisor of $p - 1$.
  - $g = h^{(p-1)/q} \bmod p$, where h is any integer with $1 < h < p - 1$ s.t. $h^{(p-1)/q} \bmod p > 1$. That is g has order $q \bmod p$.
  - *Hash* is a 1-way hash function
- **Alice's private key = x, chosen at random from $[1, q - 1]$**
- **Alice's public key = (y, p, q, g), where $y = g^x \bmod p$**

- **Signature on m is a pair (r, s)**
  - $r = g^k \bmod p$, where k is chosen at random from $[1, q - 1]$.
  - $c = Hash(r, m)$.
  - $s = (k + c * x) \bmod q$.
- **Verification of (m, r, s)**
  - Check if r is identical to $g^s * y^{-Hash(r,m)} \bmod p$

Advantage:
- Support batch verification by computing Hash(*r*,*m*) without exponentiation,
- Support multi-signature,
- EC version is adopted by Bitcoin in Taproot upgrade.

Multi-signatures

- **Authorization by multiple users is required to create a valid document**

NOTE: Projects involving Centers must have the signature of the Department Chair and Dean of the Principal Investigator's primary fa appointment. The submitting unit, in accordance with number 1 on page 1, should normally be the primary appointment of the Princ Investigator, not a Center.

x _____ 2/20/2019
Program Director/Principal Investigator — Date
PI's signature is required. No "per" signatures allowed.

For NIH projects with Multiple Principal Investigators, this should be the **Contact PI**.

John Smith _____
Printed Name

x _____ 2/20/2019
Program Director/Principal Investigator — Date

For NIH projects with Multiple PD/PI's, NIH requires signature from all individuals serving in this role (PD/PI). This is not required of Co-Investigators.

x _____ _____
Program Director/Principal Investigator — Date

x _____ 2/20/2019
Division Director — Date

If required by Department.

x _____ 2/20/2019
Facilities & Planning — Date

Facilities & Planning (as applicable per #11)

x _____ 2/20/2019
Department Chair of Primary Faculty Appointment of PI/PD (or Dean if the School of Nursing) — Date

The Department Chair of the primary faculty appointment of the principal investigator approves submissions of the project, including new space or renovation and new personnel, and certifies that new equipment requested does not unnecessarily duplicate existing resources.

_Michael Wilson_____
Printed Name: Department Chair

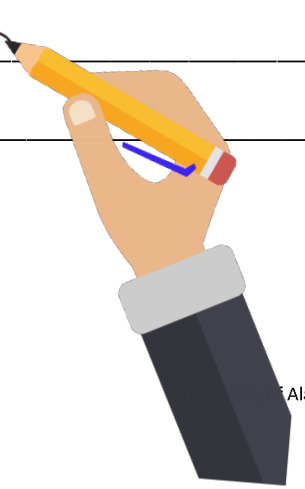
This section below is applicable if the project involves any of the following (please check all that apply):

☐ Cost Sharing (attach the cost sharing commitment form) ☐ CME / CPE ☐ New Faculty Positions ☐ Additional Space ☐ Transfer O

x _____ 2/20/2019
Dean — Date

Bob Palazzo
Printed Name

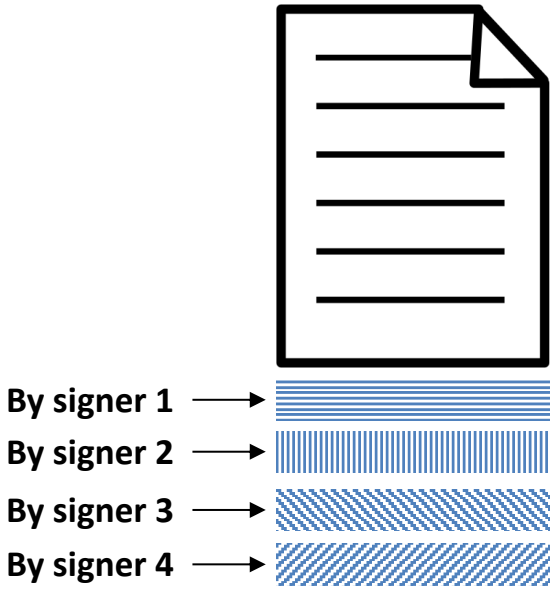
The Dean approves submissions of the project including new programs, space, and renovation and assures that: a) new programs requiring approval of Provost, the President or the Chancellor have been submitted for appropriate review and approval; b) new space requirements have received approval of the Provost; and c) agreement between or among the appropriate deans has been reached if collaboration among schools requiring sharing of indirect cost reimbursement is involved.



Multi-signatures

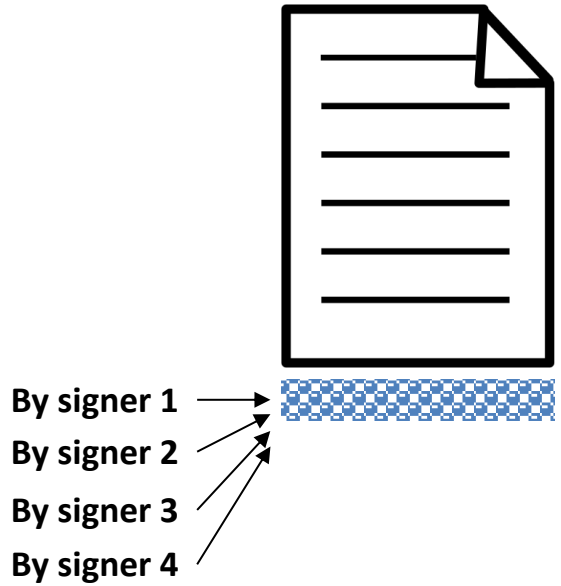
Straightforward solution

- Each signer signs the document independently
  ➔ n signatures



Better solution

- Signers collaboratively generate a compact signature
  ➔ same size as a single signature



Schnorr Multi-Signature (MuSig)
-- System wide common parameters --

- System wide: $(p, q, g, H_{com}, H_{agg}, H_{sig})$,
  where
  - p = a prime of at least 2048 bits.
  - q = a 224-bit prime divisor of $p - 1$.
  - $g = h^{(p-1)/q} \bmod p$, where h is any integer with $1 < h < p - 1$ s.t.
    $h^{(p-1)/q} \bmod p > 1$.
    That is g has order $q \bmod p$.

  - H_{com}: 1-way hash for commitment purpose.
  - H_{agg}: 1-way hash for public key aggregation.
  - H_{sig}: 1-way hash for signature.
  - (In practice the 3 hash algorithms could be the same but prefixed with different tags in input)

Schnorr Multi-Signature – Users' private & public keys

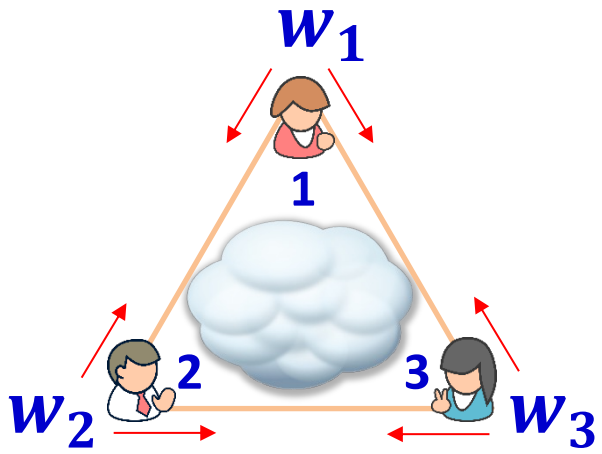
- User i:
  - Private key x_i:
    - an integer randomly selected from $[1, q-1]$.
  - Public key y_i:
  - $y_i = g^{x_i} \bmod p$

- Users 1, 2, …, n
  - Ordered list of all public keys: $L = \{y_1, y_2, \ldots, y_n\}$
  - "aggregate" public key
    - $Y_{agg} = \prod_{j=1}^{n} y_j^{a_j} \bmod p$,

      where $a_j = H_{agg}(L, y_j)$

Signature generation Round 1 (broadcast commitments $w_1, w_2, \ldots, w_n$)

1. Pick $k_1 \overset{U}{\leftarrow} [1, q-1]$,
2. Compute $r_1 = g^{k_1} (\mathbf{mod}\ p)$,
3. Compute $w_1 = H_{com}(r_1)$,
4. Broadcast w_1 to other co-signers,
5. Wait to receive w_2 and w_3 from other co-signers,



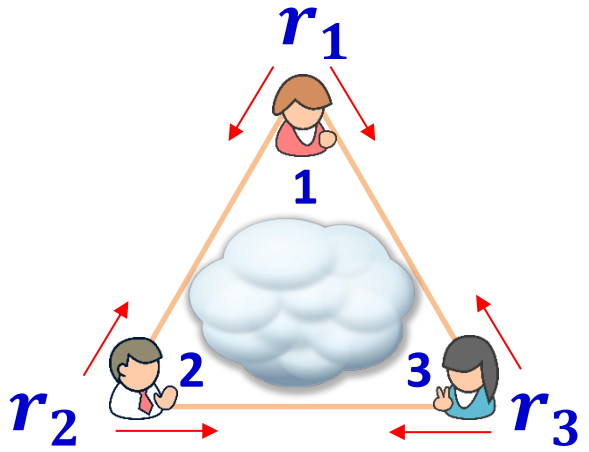
1. Pick $k_2 \overset{U}{\leftarrow} [1, q-1]$,
2. Compute $r_2 = g^{k_2} (\mathbf{mod}\ p)$,
3. Compute $w_2 = H_{com}(r_2)$,
4. Broadcast w_2 to other co-signers,
5. Wait to receive w_1 and w_3 from other co-signers,

1. Pick $k_3 \overset{U}{\leftarrow} [1, q-1]$,
2. Compute $r_3 = g^{k_3} (\mathbf{mod}\ p)$,
3. Compute $w_3 = H_{com}(r_3)$,
4. Broadcast w_3 to other co-signers,
5. Wait to receive w_1 and w_2 from other co-signers,

Signature generation Round 2

(broadcast random powers $r_1, r_2, \ldots, r_n$)

6. Broadcast r_1 to other co-signers,
7. Wait to receive r_2 and r_3 from other co-signers,
8. Proceed only if $w_j = H_{com}(r_j)$, for both $j \in \{2, 3\}$,



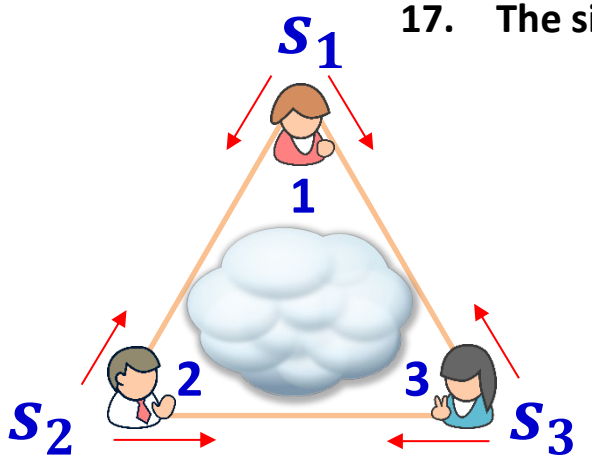
6. Broadcast r_2 to other co-signers,
7. Wait to receive r_1 and r_3 from other co-signers,
8. Proceed only if $w_j = H_{com}(r_j)$, for both $j \in \{1, 3\}$,

6. Broadcast r_3 to other co-signers,
7. Wait to receive r_1 and r_2 from other co-signers,
8. Proceed only if $w_j = H_{com}(r_j)$, for both $j \in \{1, 2\}$,

Signature generation Round 3

(broadcast partial signatures $s_1, s_2, \ldots, s_n$)

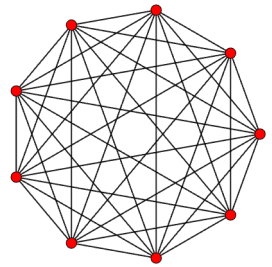
9. Compute $a_j = H_{agg}(L, y_j)$, for $j = 1, 2, 3$
10. Compute $Y_{agg} = \prod_{i=1}^{3} y_i^{a_i} \pmod p$,
11. Compute $R = \prod_{j=1}^{3} r_j \pmod p$,
12. Compute $c = H_{sig}(Y_{agg}, R, m)$,
13. Compute $s_1 = k_1 + ca_1x_1 \pmod q$,
14. Broadcast s_1 to other co-signers,
15. Wait to receive s_2 and s_3 from other co-signers,
16. Compute $s = \sum_{j=1}^{3} s_j \pmod q$,
17. The signature is $\sigma = (R, s)$.



9. Compute $a_j = H_{agg}(L, y_j)$, for $j = 1, 2, 3$
10. Compute $Y_{agg} = \prod_{i=1}^{3} y_i^{a_i} \pmod p$,
11. Compute $R = \prod_{j=1}^{3} r_j \pmod p$,
12. Compute $c = H_{sig}(Y_{agg}, R, m)$,
13. Compute $s_2 = k_2 + ca_2x_2 \pmod q$,
14. Broadcast s_2 to other co-signers,
15. Wait to receive s_1 and s_3 from other co-signers,
16. Compute $s = \sum_{j=1}^{3} s_j \pmod q$,
17. The signature is $\sigma = (R, s)$.

9. Compute $a_j = H_{agg}(L, y_j)$, for $j = 1, 2, 3$
10. Compute $Y_{agg} = \prod_{i=1}^{3} y_i^{a_i} \pmod p$,
11. Compute $R = \prod_{j=1}^{3} r_j \pmod p$,
12. Compute $c = H_{sig}(Y_{agg}, R, m)$,
13. Compute $s_3 = k_3 + ca_3x_3 \pmod q$,
14. Broadcast s_3 to other co-signers,
15. Wait to receive s_1 and s_2 from other co-signers,
16. Compute $s = \sum_{j=1}^{3} s_j \pmod q$,
17. The signature is $\sigma = (R, s)$.

Schnorr multi-signature
– signing m by each user i in {1,2,...,n} –



1

1. Pick $k_i \overset{U}{\leftarrow} [1, q-1]$,
2. Compute $r_i = g^{k_i} (\mathbf{mod}\ p)$,
3. Compute $w_i = H_{com}(r_i)$,
4. Broadcast w_i to all other co-signers,
5. Wait to receive w_j from all other co-signers,

2

6. Broadcast r_i to all other co-signers,
7. Wait to receive r_j from all other co-signers,
8. Proceed only if $w_j = H_{com}(r_j)$, for all $j = 1, 2, \dots, n$,

3

9. Compute $a_j = H_{agg}(L, y_j)$, for all $j = 1, 2, \dots, n$,
10. Compute $Y_{agg} = \prod_{j=1}^{n} y_j^{a_j} \ (\mathbf{mod}\ p)$,
11. Compute $R = \prod_{j=1}^{n} r_j \ (\mathbf{mod}\ p)$,
12. Compute $c = H_{sig}(Y_{agg}, R, m)$,
13. Compute $s_i = k_i + c a_i x_i \ (\mathbf{mod}\ q)$,
14. Broadcast s_i to all other co-signers,
15. Wait to receive s_j from all other co-signers,
16. Compute $s = \sum_{j=1}^{n} s_j \ (\mathbf{mod}\ q)$,
17. The signature is $\sigma = (R, s)$.

$$\boldsymbol{\sigma = (R, s)}$$

- **Note that**
  - $R = \prod_{j=1}^{n} r_j \bmod p = \prod_{j=1}^{n} g^{k_j} \bmod p = g^{\sum_{j=1}^{n} k_j} \bmod p$
  - $s = \sum_{j=1}^{n} s_j \bmod q = \sum_{j=1}^{n} (k_j + c a_j x_j) \bmod q$

$$= \sum_{j=1}^{n} k_j + c \sum_{j=1}^{n} (a_j x_j) \bmod q$$

- **Further note the aggregate public key Y_{agg}:**
  - $Y_{agg} = \prod_{j=1}^{n} y_j^{a_j} \bmod p = g^{\sum_{j=1}^{n} (a_j x_j)} \bmod p$, where $a_j = H_{agg}(L, y_j)$.

Schnorr Multi-Signature – verification

- To verify (m, R, s) using Y_{agg}
  - Get the ordered list of all public keys:
    $L = \{y_1, y_2, \dots, y_n\}$
  - Compute the aggregate public key
    $Y_{agg} = \prod_{j=1}^{n} y_j^{a_j} \bmod p$, where $a_j = H_{agg}(L, y_j)$
  - Compute $c = H_{sig}(R, Y_{agg}, m)$,
  - Accept the signature only if
    $$R = g^s Y_{agg}^{-c} \bmod p$$

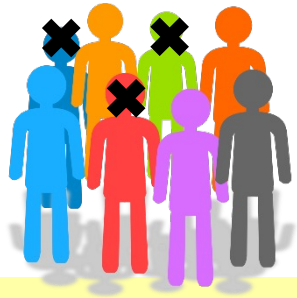
Indistinguishability

	Single party signature	Multi-signature	Note
Signature	(r, s)	(R, s)	Same size
Public key	y (for user i)	$Y_{agg} = \prod_{j=1}^{n} y_j^{a_j}$ (aggregate)	Same size
Verification	**Check if** $r = g^s * y^{-Hash(r, y_i, m)} \bmod p$	**Check if** $R = g^s * Y_{agg}^{-H_{sig}(R, Y_A, m)} \bmod p$	Identical (choose $H_{sig} = Hash$)

Schnorr (t, n) Threshold Signature

- **Goal**
  - **At least t out of n users are required to collectively create a valid signature**
  - **Signature is compact: same size as a single Schnorr signature**
  - **Can be verified by anyone**
- **Let $T \in \{1, 2, \dots, t\}$ be a subset of t users.**
- **Let $L = \{y_1, y_2, \dots, y_n\}$ be the ordered set of all n users' public keys**
- **Let L_T be the ordered set of public keys of all t users' in T.**

Schnorr (t, n) threshold signature – signing m by each user i in T –



$n - t$ users unavailable or sitting out

1

1. Pick $k_i \overset{U}{\leftarrow} [1, q-1]$,
2. Compute $r_i = g^{k_i} (\mathbf{mod}\ p)$,
3. Compute $w_i = H_{com}(r_i)$,
4. Broadcast w_i to all other co-signers in T,
5. Wait to receive w_j from all other co-signers in T,

2

6. Broadcast r_i to all other co-signers in T,
7. Wait to receive r_j from all other co-signers in T,
8. Proceed only if $w_j = H_{com}(r_j)$, for all $j \in T$,

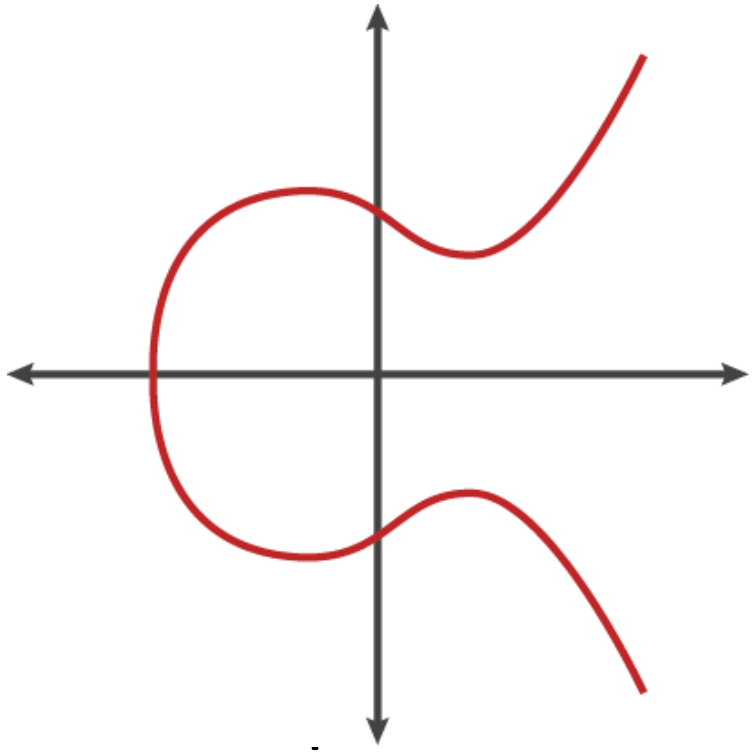
3

9. Compute $a_j = H_{agg}(L_T, y_i)$, for all $j \in T$,
10. Compute $Y_{agg} = \prod_{j \in T} y_j^{a_j} (\mathbf{mod}\ p)$,
11. Compute $R = \prod_{j \in T} r_j (\mathbf{mod}\ p)$,
12. Compute $c = H_{sig}(Y_{agg}, R, m)$,
13. Compute $s_i = k_i + c a_i x_i (\mathbf{mod}\ q)$,
14. Broadcast s_i to all other co-signers in T,
15. Wait to receive s_j from all other co-signers in T,
16. Compute $s = \sum_{j \in T} s_j \ (\mathbf{mod}\ q)$,
17. The signature is $\sigma = (R, s)$.

Schnorr (t, n) Threshold Signature – Verification –

- To verify (m, R, s) for a t-subset T
  - Get the ordered list of all public keys $L = \{y_1, y_2, \ldots, y_n\}$
  - From L construct L_T, the ordered set of public keys of all t users in T.
    - If L_T, instead of T, is given, make sure $L_T \subseteq L$; Derive T from L_T.
  - Compute the aggregate public key
    $Y_{agg} = \prod_{j \in T} y_j^{a_j} \bmod p$, where $a_j = H_{agg}(L_T, y_i)$
  - Compute $c = H_{sig}(R, Y_{agg}, m)$,
  - Accept the signature only if $R = g^s Y_{agg}^{-c} \bmod p$.

Elliptic Curve Cryptosystems



Why use Elliptic Curves?

- **Short answer:**
  - **faster and more efficient than competition**
- **Long answer:**
  - **The best *currently known* algorithm for EC discrete logarithms would take about as long to find a 160-bit EC discrete log as the best *currently known* algorithm for integer discrete logarithms would take to find a 1024-bit discrete log.**
  - **160-bit EC algorithms are somewhat faster and use shorter keys than 1024-bit "traditional" algorithms.**

Elliptic curve

- For a prime $p > 3$, an elliptic curve E defined over Z_p takes the form of

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

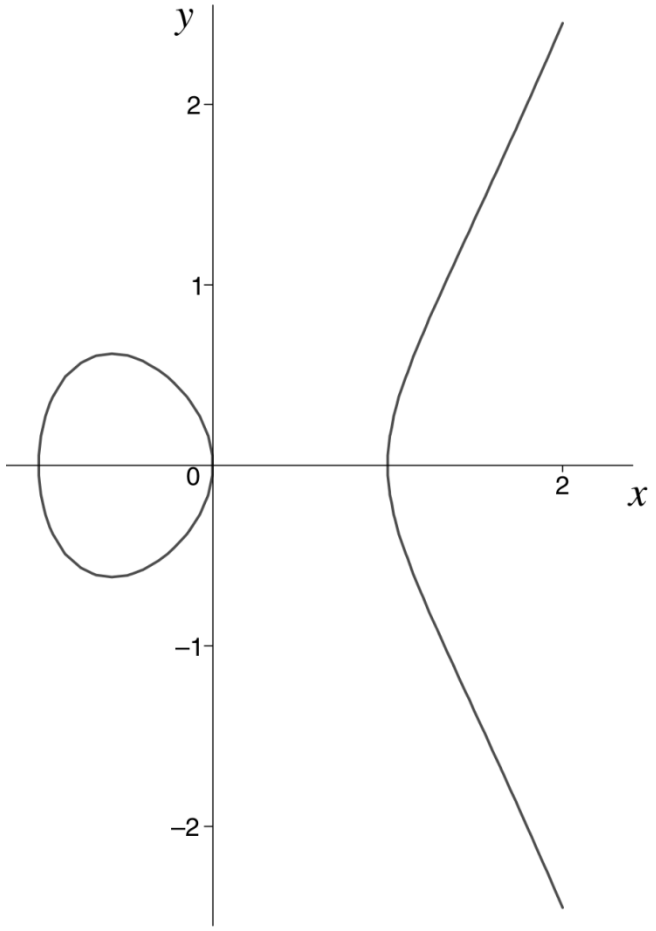
  where
  - a and b are integers modulo p
  - $4a^3 + 27b^2 \neq 0 \pmod{p}$.
- Note: to be precise, it should be GF(p) rather than Z_p.

Visualizing elliptic curves

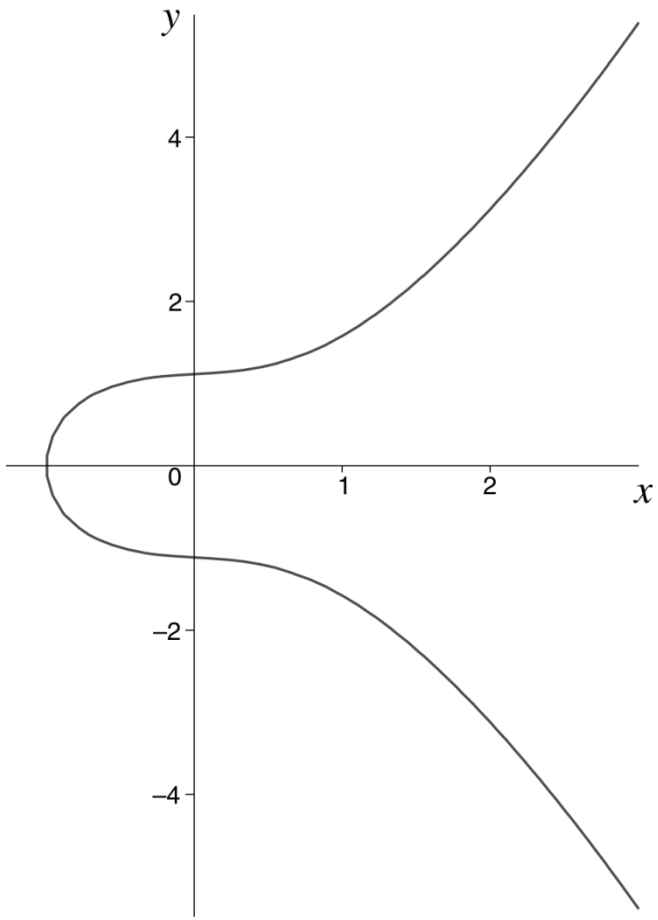
- **Elliptic curves over finite fields are discrete, algebraic objects --- they cannot be plotted like their counterparts over the real.**

- **However, some may find it helpful to look at the plotting of elliptic curves over the real.**

Elliptic curves over the real



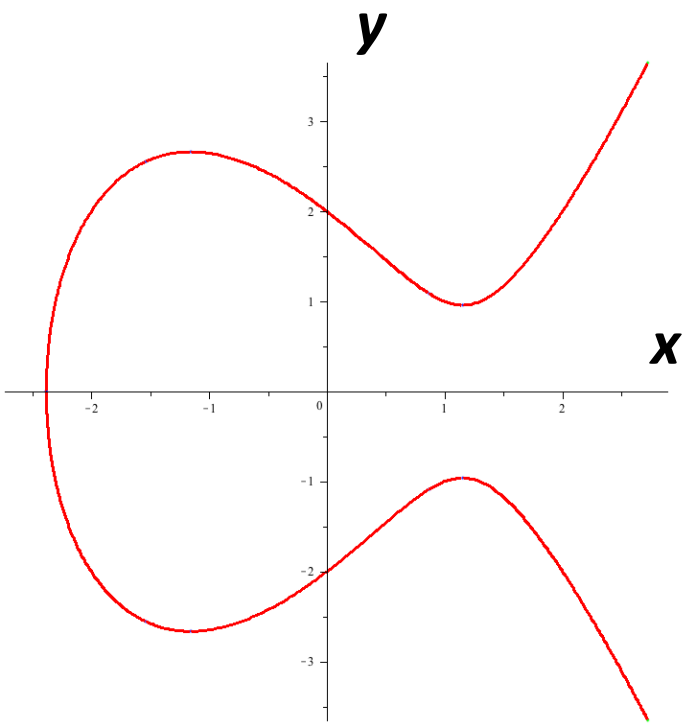
(a) $E_1 : y^2 = x^3 - x$



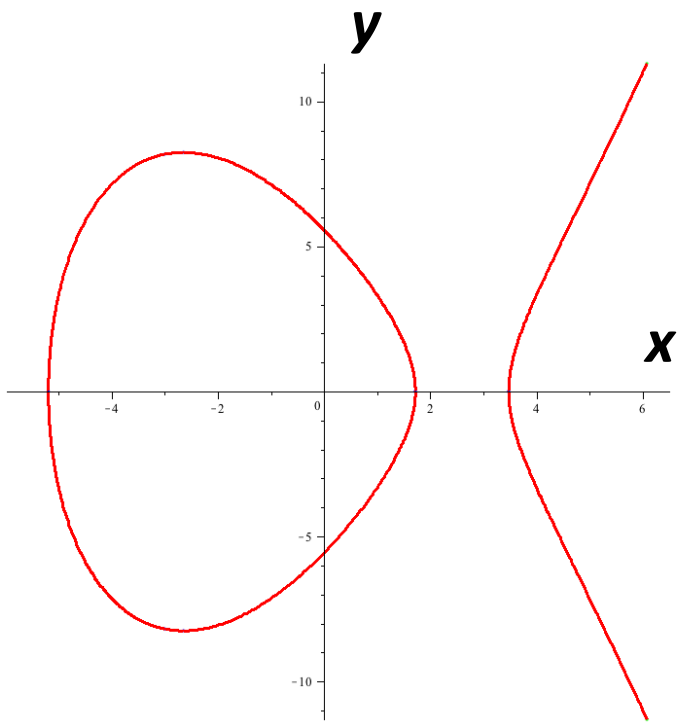
(b) $E_2 : y^2 = x^3 + \frac{1}{4}x + \frac{5}{4}$

More elliptic curves over the real

$$y^2 = x^3 - 4x + 4$$

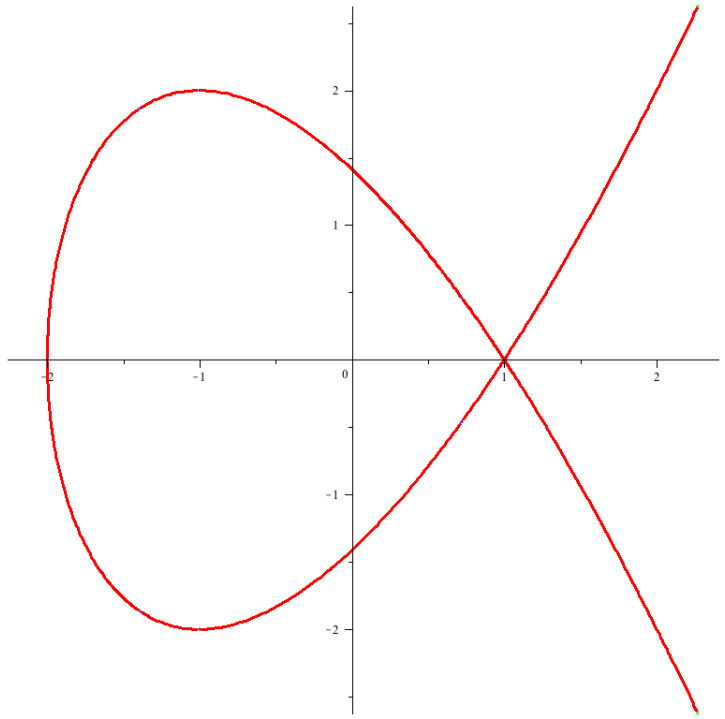


$$y^2 = x^3 + 21x - 31$$

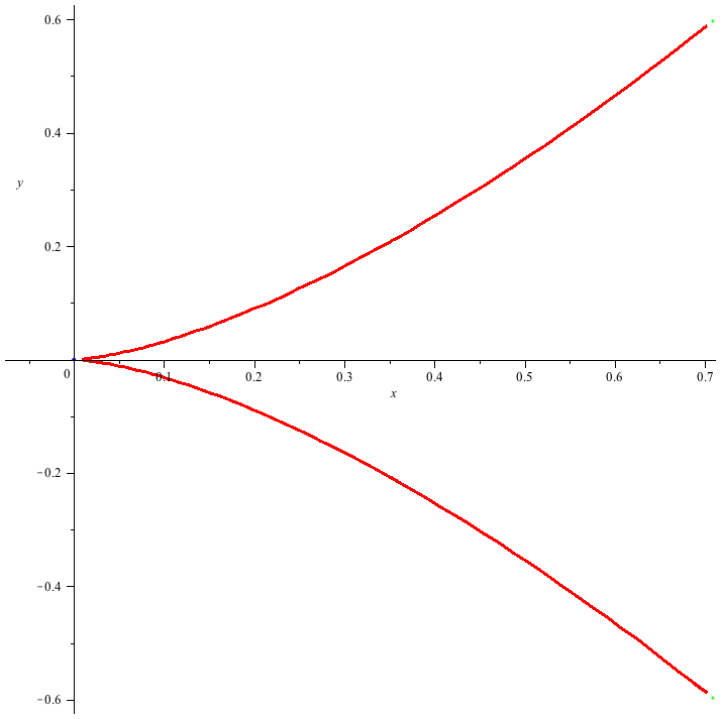


Yet more

$$y^2 = x^3 - 3x + 2$$



$$y^2 = x^3$$



Points on elliptic curve

- Let E be the curve

$$y^2 = x^3 + 10x + 5$$

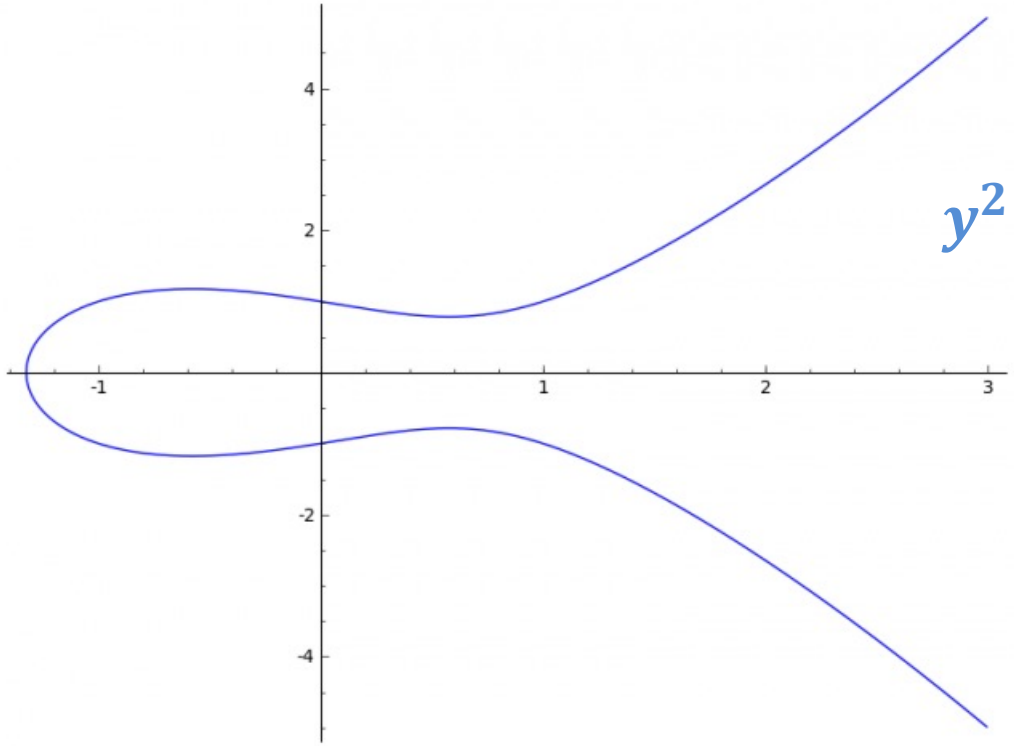
over Z_{13}.

- Then there are 10 points on E:

```
{ 0,    (1,4), ( 1,9), ( 3,6), ( 3,7),
 (8,5), (8,8), (10,0), (11,4), (11,9)}
```

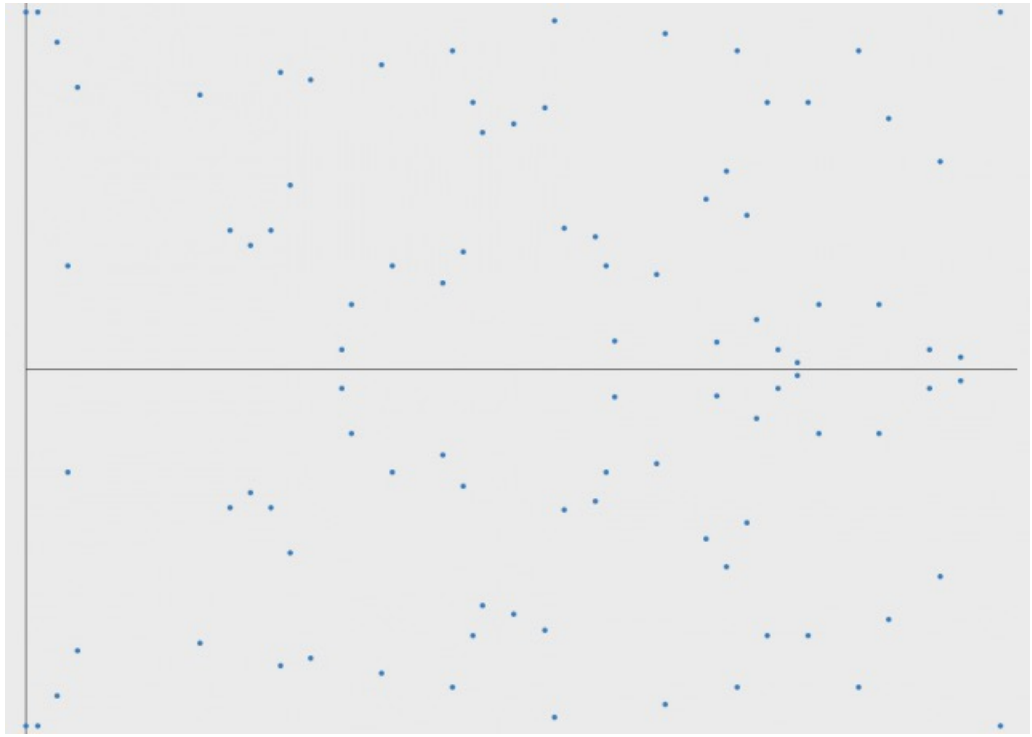
where 0 is called the point-at-infinity.

Curves are an illusion!



$$y^2 = x^3 - x + 1$$

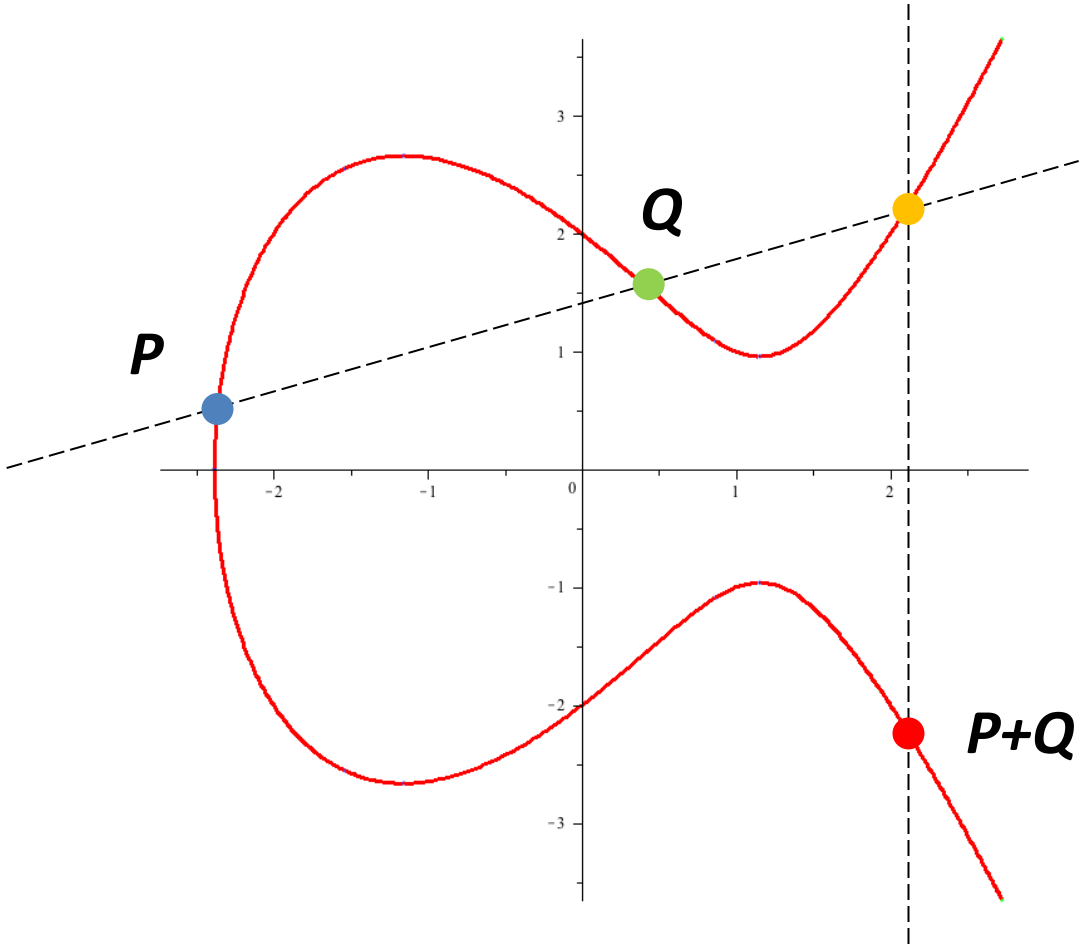
$$y^2 = x^3 - x + 1 \text{ over } Z_{97}$$



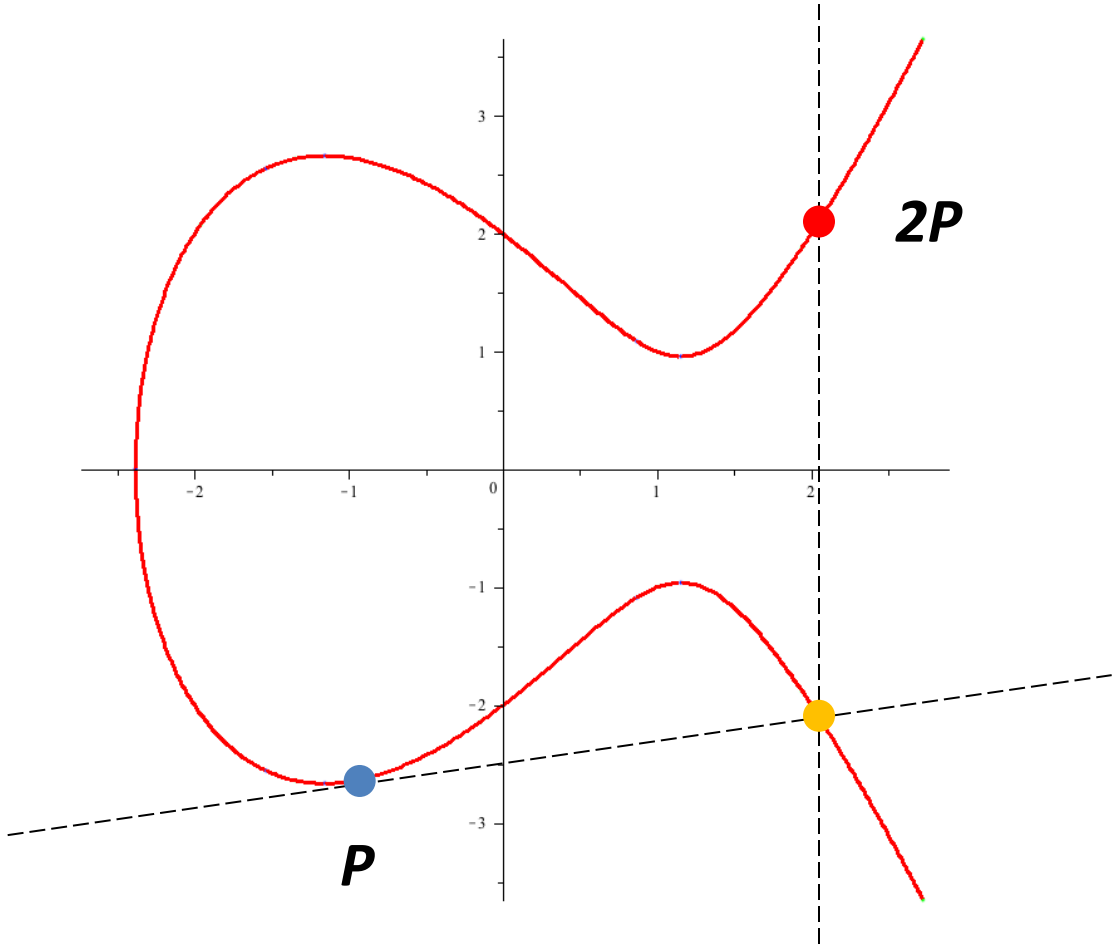
Points on elliptic curve

- It turns out points on an elliptic curve E over Z_p form a group under an appropriately defined group operation "+"
  - Addition "+" of 2 points on an elliptic curve has a geometric interpretation:
    it follows the "chord-and-tangent" rule
- This group of points on E over Z_p is denoted by $E(Z_p)$.

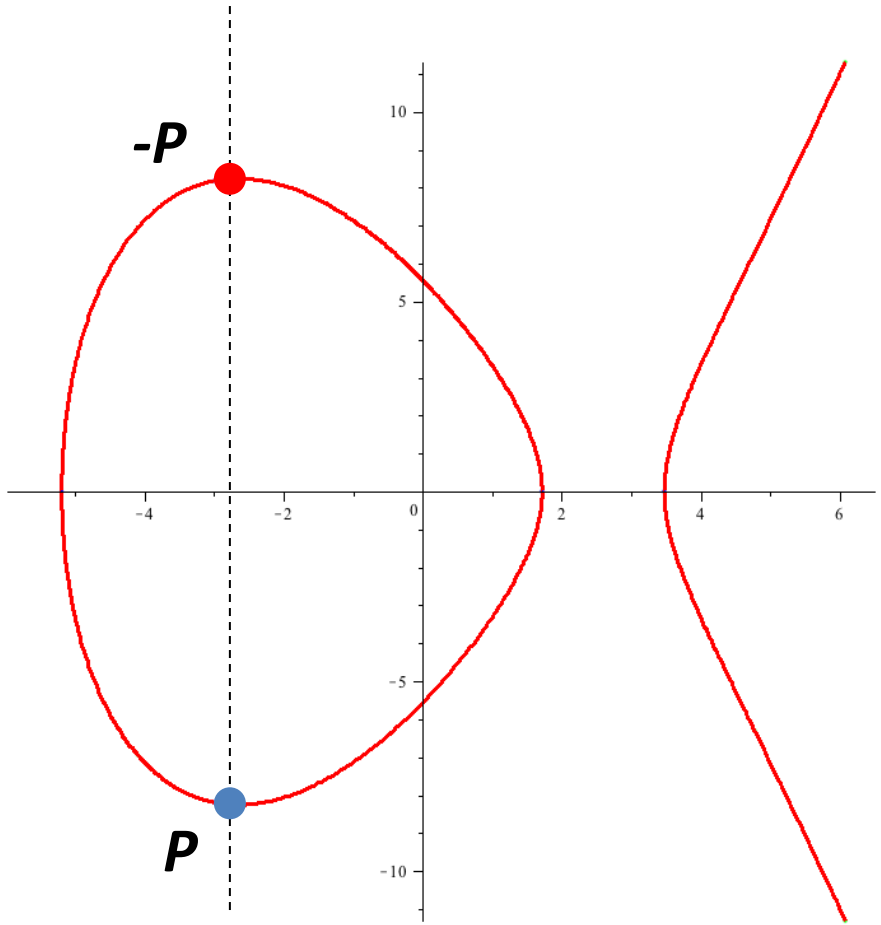
Addition by the chord rule: *P*+*Q*



Doubling by the tangent rule: 2*P*=*P*+*P*



Additive inverse: -*P*



Formula for computing *P*+*Q* and 2*P*

- Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are 2 points on an elliptic curve over Z_p where $P \neq -Q$.

- Then $P + Q = (x_3, y_3)$ can be computed by
  - $x_3 = \lambda^2 - x_1 - x_2 \pmod p$
  - $y_3 = \lambda(x_1 - x_3) - y_1 \pmod p$

  where

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \bmod p, & \text{if } P \neq Q \\ \frac{3x_1^2 + a}{2y_1} \bmod p, & \text{if } P = Q \end{cases}$$

Example

- Let p = 23 and consider the elliptic curve $y^2 = x^3 + x + 1$ defined over Z_{23}.
  Note we have $a = 1$ and $b = 1$, and that
  $4a^3 + 27b^2 = 4 + 4 = 8 \neq 0$;
  so E is indeed an elliptic curve.

- There are 28 points over the elliptic curve, including 0 the *point at infinity*.

0 (point at infinity)		
(0, 1)	(6, 4)	(12, 19)
(0, 22)	(6, 19)	(13, 7)
(1, 7)	(7, 11)	(13, 16)
(1, 16)	(7, 12)	(17, 3)
(3, 10)	(9, 7)	(17, 20)
(3, 13)	(9, 16)	(18, 3)
(4, 0)	(11, 3)	(18, 20)
(5, 4)	(11, 20)	(19, 5)
(5, 19)	(12, 4)	(19, 18)

Example (cnt'd)

- Let $P = (3, 10)$ and $Q = (9, 7)$.
  Then $P + Q = (x_3, y_3)$ is computed as follows:
  - $\lambda = \frac{y_2 - y_1}{x_2 - x_1} = \frac{-3}{6} = 20 * 4 = 11 \pmod{23}$
  - $x_3 = 11^2 - 3 - 9 = 6 - 3 - 9 = -6 = 17 \pmod{23}$, and
  - $y_3 = 11(3 - (-6)) - 10 = 11(9) - 10 = 89 = 20 \pmod{23}$.

  Hence $P + Q = (17, 20)$.

- Let $P = (3, 10)$.
  Then $2P = P + P = (x_3, y_3)$ is computed as follows:
  - $\lambda = \frac{3 * x_1^2 + 1}{2y_1} = \frac{3 * 3^2 + 1}{2 * 10} = 5 * 15 = 6 \pmod{23}$
  - $x_3 = 6^2 - 6 = 30 = 7 \pmod{23}$, and
  - $y_3 = 6(3 - 7) - 10 = -24 - 10 = -11 = 12 \pmod{23}$.

  Hence $2P = (7, 12)$.

Base point / generator

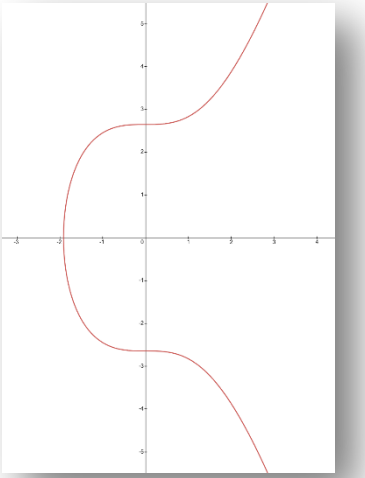
- The group defined by an elliptic curve E may have sub-groups.
- The size of a sub-group must be a factor of the size of the group defined by E.
- Each sub-group has its own generators, or base points, that generates the sub-group.

- E.g. consider the same elliptic curve
  $y^2 = x^3 + x + 1$ defined over Z_{23}
  - As there are 28 (=2*2*7) points on the curve, the order of a point must be 1,2,7, or 28.
  - Consider $G = (3, 10)$. Since $2G = (7, 12) \neq 0$ and $7G = (11, 3) \neq 0$, G must have the max. order 28. That is it's a base point/generator for the elliptic curve.

ECDSA: Elliptic Curve Digital Signature Algorithm

(Used in Bitcoin)

Elliptic curve used in Bitcoin: secp256k1



- (E, G, p, n) are system-wide parameters, used by all parties in Bitcoin
  - E: an elliptic curve on GF(p)
  - G: a base point
  - p: a prime number
  - n: size or order of the additive group of points on E
  - (For both ECDSA and Schnorr)
- The elliptic curve E (a Koblitz curve): $y^2 = x^3 + 7 \pmod p$ over $GF(p)$ with the following parameters (in hex format):
  - p: a prime number

    $p = FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFE\ FFFFFC2F$

    $= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$
  - $b = 7$
  - Base point $G = (x_G, y_G)$
    - $x_G = 79BE667E\ F9DCBBAC\ 55A06295\ CE870B07\ 029BFCDB\ 2DCE28D9\ 59F2815B\ 16F81798$
    - $y_G = 483ADA77\ 26A3C465\ 5DA4FBFC\ 0E1108A8\ FD17B448\ A6855419\ 9C47D08F\ FB10D4B8$
  - $n = FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFE\ BAAEDCE6\ AF48A03B\ BFD25E8C\ D0364141$ (a prime order)

ECDSA & Schnorr key pair generation: (d, Q)

- User specific
- Alice does the following:
  - Select, uniformly at random, an integer d from [2, n-2].
  - Compute $Q = dG$.
  - Alice's public-private key pair
    - public key is Q, together with system wide (E, G, p, n), 1-way hash etc;
    - private signing key is d.

ECDSA signature generation & verification

Signature generation

- To sign a message m, Alice does the following:
  1. Select uniformly at random an integer k in the interval [2, n-2].
  2. Compute $kG = (x_1, y_1)$ and $r = x_1 \pmod n$.
     If $r = 0$, then go to step 1.
  3. Compute $s = (\mathrm{Hash}(m) + d \cdot r)/k \pmod n$, where Hash is a 1-way hash.
  4. If $s = 0$, then go to step 1.
     Otherwise, the signature for the message m is the pair of integers $\sigma=(r, s)$.

Signature verification

- To verify Alice's signature $\sigma= (r, s)$ on m, Bob performs the following:
  1. Obtain an authentic copy of Alice's public key (E, G, p, n, Q).
  2. Verify that r and s are non-zero integers smaller than n.
  3. Compute $w = s^{-1} \pmod n$.
  4. Compute $u_1 = \mathrm{Hash}(m) \cdot w \pmod n$, $u_2 = r \cdot w \pmod n$.
  5. Compute $u_1 G + u_2 Q = (x_1, y_1)$
  6. Accept the signature only if $x_1 \pmod n = r$.

Malleability of ECDSA

- If (r, s) is a valid signature of m,
  then both $(r, -s \bmod n)$, and $(r, n - s)$ are a valid signature of m also.
- Reason:
  r depends on the x coordinate of $kG = (x_1, y_1)$ only.
  Therefore changing kG to $-kG$ flips the sign of its y coordinate only, having no affect on the value of r. (Note also the slide on "Additive Inverse" of a point on EC.)
- As a result, $(r, -s \bmod n)$ will pass signature verification:
  - $(r, -s \bmod n)$ ➔ $(-u_1)G + (-u_2)Q = -(u_1G + u_2Q) = (x_0, -y_0)$
    ➔ the same $v = x_0 \pmod n$

- This is one of the root causes of Bitcoin "transaction malleability".
- Remedy:
  r should be a function of both coordinates of $kG = (x_1, y_1)$.
  E.g. $r = \mathrm{Hash}(x_1, y_1)$

EC-Schnorr signature: (r, s) version

- To sign a message m, Alice does the following:
  1. Select k uniformly at random from [2, n-2].
  2. Compute $r = kG$.
  3. Compute $c = \mathrm{Hash}(r, m)$.
  4. Compute $s = k + c \cdot d \pmod{n}$
  5. If either e or s is zero, go back to Step 1. Otherwise the signature for m is σ=(r, s).

- To verify (m, r, s), Bob performs the following:
  1. Get Alice's public key (E, G, p, n, Q).
  2. Verify that s is non-zero & both r and s are within correct ranges.
  3. Compute a candidate $r' = sG - \mathrm{Hash}(r, m)Q$
  4. Output "valid" iff $r' = r$

Why EC-Schnorr is better?

- Admit
  - Security proofs
  - Easy to implement threshold signature
  - Blind signature
- Free of signature malleability
- Cleaner and slightly faster
  - No inversion in verification

EC-SCHNORR MULTI-SIGNATURE (EC-MUSIG)

Multi-signatures

- **Authorization by multiple users is required to create a valid document**

NOTE: Projects involving Centers must have the signature of the Department Chair and Dean of the Principal Investigator's primary fa appointment. The submitting unit, in accordance with number 1 on page 1, should normally be the primary appointment of the Princ Investigator, not a Center.

x _____ 2/20/2019
Program Director/Principal Investigator — Date
PI's signature is required. No "per" signatures allowed.

For NIH projects with Multiple Principal Investigators, this should be the **Contact PI**.

John Smith _____
Printed Name

x _____ 2/20/2019
Program Director/Principal Investigator — Date

For NIH projects with Multiple PD/PI's, NIH requires signature from all individuals serving in this role (PD/PI). This is not required of Co-Investigators.

x _____ _____
Program Director/Principal Investigator — Date

x _____ 2/20/2019
Division Director — Date

If required by Department.

x _____ 2/20/2019
Facilities & Planning — Date

Facilities & Planning (as applicable per #11)

x _____ 2/20/2019
Department Chair of Primary Faculty Appointment of PI/PD (or Dean if the School of Nursing) — Date

The Department Chair of the primary faculty appointment of the principal investigator approves submissions of the project, including new space or renovation and new personnel, and certifies that new equipment requested does not unnecessarily duplicate existing resources.

_Michael Wilson_____
Printed Name: Department Chair

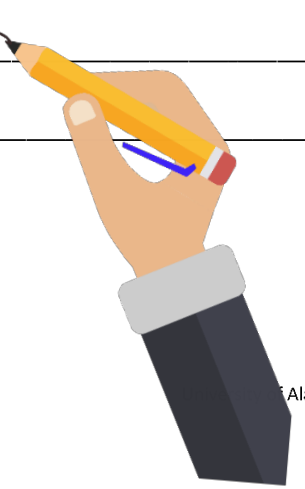
This section below is applicable if the project involves any of the following (please check all that apply):

☐ Cost Sharing (attach the cost sharing commitment form) ☐ CME / CPE ☐ New Faculty Positions ☐ Additional Space ☐ Transfer O

x _____ 2/20/2019
Dean — Date

Bob Palazzo
Printed Name

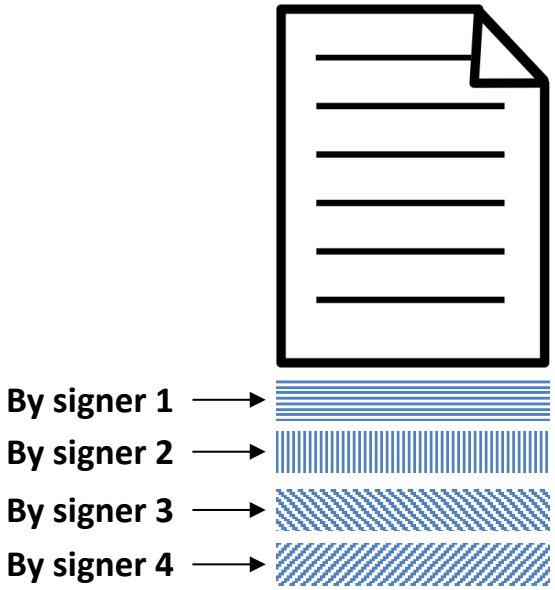
The Dean approves submissions of the project including new programs, space, and renovation and assures that: a) new programs requiring approval of Provost, the President or the Chancellor have been submitted for appropriate review and approval; b) new space requirements have received approval of the Provost; and c) agreement between or among the appropriate deans has been reached if collaboration among schools requiring sharing of indirect cost reimbursement is involved.



Multi-signatures

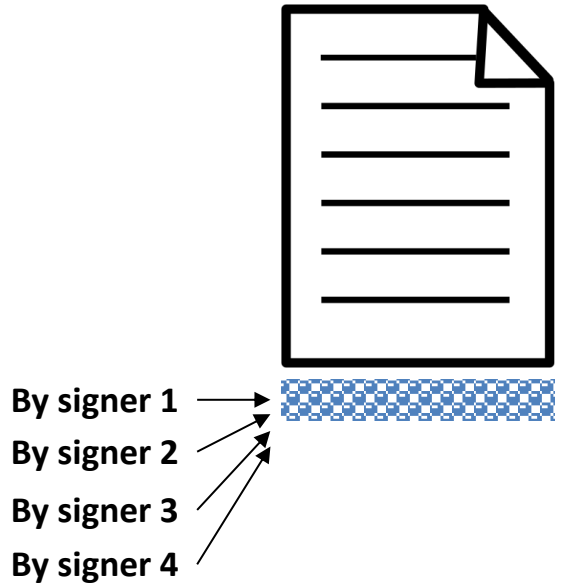
Straightforward solution

- Each signer signs the document independently
  ➔ n signatures



Better solution

- Signers collaboratively generate a compact signature
  ➔ same size as a single signature



EC-Schnorr Multi-Signature
-- System wide common parameters --

- **Elliptic curve & 1-way hash algorithms:**
  $$(E, G, p, n, H_{com}, H_{agg}, H_{sig}),$$
  where
  - (E, G, p, n) **specify an elliptic curve, e.g. secp256k1**
    - E: an elliptic curve on GF(p)
    - G: a base point
    - p: a prime number
    - n: size of the additive group of points on E
  - H_{com}: 1-way hash for commitment purpose.
  - H_{agg}: 1-way hash for public key aggregation.
  - H_{sig}: 1-way hash for signature.
  - (In practice the 3 hash algorithms could be the same but prefixed with different tags in input)

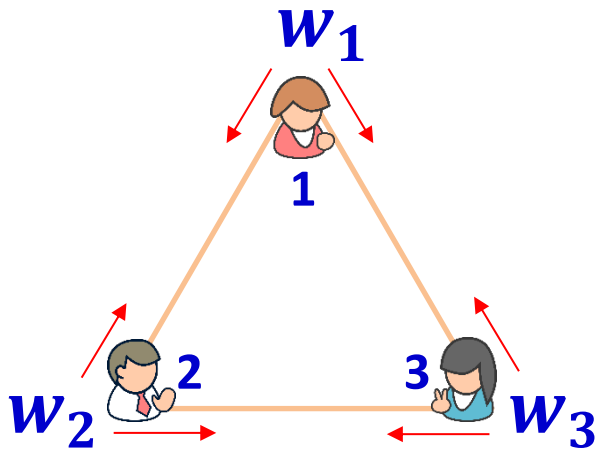
EC-Schnorr Multi-Signature – Users' private & public keys

- u users: $1, 2, \ldots, u$
- Public-private key pair for user i:
  - Public key y_i: $Q_i = d_i G$
  - Private key d_i:
    - an integer randomly selected from $[2, n-2]$.
- Users $1, 2, \ldots, u$
  - Ordered list of all public keys: $L = \{Q_1, Q_2, \ldots, Q_u\}$
  - "aggregate" public key
    - $Q_{agg} = \sum_{j=1}^{u} a_j Q_j$,

      where $a_j = H_{agg}(L, Q_j)$

Signature generation Round 1 (broadcast w_1, w_2, w_3)

1. Pick k_1 at random from $[2, n-2]$,
2. Compute $r_1 = k_1 G$,
3. Compute $w_1 = H_{com}(r_1)$,
4. Broadcast w_1 to other co-signers,
5. Receive w_2 and w_3 from other co-signers,

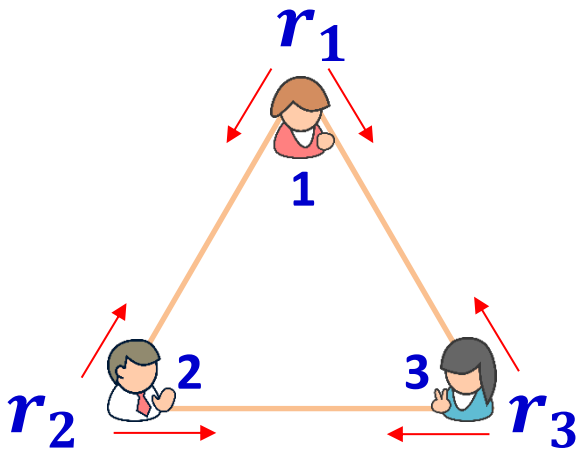


1. Pick k_2 at random from $[2, n-2]$,
2. Compute $r_2 = k_2 G$,
3. Compute $w_2 = H_{com}(r_2)$,
4. Broadcast w_2 to other co-signers,
5. Receive w_1 and w_3 from other co-signers,

1. Pick k_3 at random from $[2, n-2]$,
2. Compute $r_3 = k_3 G$,
3. Compute $w_3 = H_{com}(r_3)$,
4. Broadcast w_3 to other co-signers,
5. Receive w_1 and w_2 from other co-signers,

Signature generation Round 2 (broadcast r_1, r_2, r_3)

6. Broadcast r_1 to other co-signers,
7. Receive r_2 and r_3 from other co-signers,
8. Proceed only if $w_j = H_{com}(r_j)$, for both $j \in \{2, 3\}$,



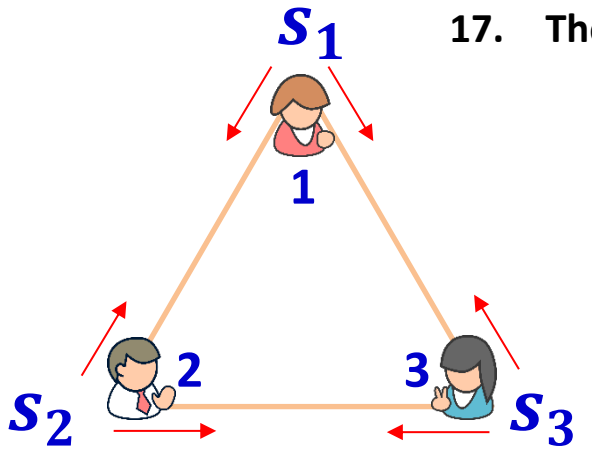
6. Broadcast r_2 to other co-signers,
7. Receive r_1 and r_3 from other co-signers,
8. Proceed only if $w_j = H_{com}(r_j)$, for both $j \in \{1, 3\}$,

6. Broadcast r_3 to other co-signers,
7. Receive r_1 and r_2 from other co-signers,
8. Proceed only if $w_j = H_{com}(r_j)$, for both $j \in \{1, 2\}$,

Signature generation Round 3

(broadcast s_1, s_2, s_3)

9. Compute $a_j = H_{agg}(L, Q_j)$, for $j = 1, 2, 3$
10. Compute $Q_{agg} = \sum_{j=1}^{3} a_j Q_j$,
11. Compute $R = \sum_{j=1}^{3} r_j$,
12. Compute $c = H_{sig}(R, Q_{agg}, m)$,
13. Compute $s_1 = k_1 + ca_1 d_1 \bmod n$,
14. Broadcast s_1 to other co-signers,
15. Receive s_2 and s_3 from other co-signers,
16. Compute $s = \sum_{j=1}^{3} s_j \bmod n$,
17. The signature is $\sigma = (R, s)$.



9. Compute $a_j = H_{agg}(L, Q_j)$, for $j = 1, 2, 3$
10. Compute $Q_{agg} = \sum_{j=1}^{3} a_j Q_j$,
11. Compute $R = \sum_{j=1}^{3} r_j$,
12. Compute $c = H_{sig}(R, Q_{agg}, m)$,
13. Compute $s_2 = k_2 + ca_2 d_2 \bmod n$,
14. Broadcast s_2 to other co-signers,
15. Receive s_1 and s_3 from other co-signers,
16. Compute $s = \sum_{j=1}^{3} s_j \bmod n$,
17. The signature is $\sigma = (R, s)$.

9. Compute $a_j = H_{agg}(L, Q_j)$, for $j = 1, 2, 3$
10. Compute $Q_{agg} = \sum_{j=1}^{3} a_j Q_j$,
11. Compute $R = \sum_{j=1}^{3} r_j$,
12. Compute $c = H_{sig}(R, Q_{agg}, m)$,
13. Compute $s_3 = k_3 + ca_3 d_3 \bmod n$,
14. Broadcast s_3 to other co-signers,
15. Receive s_1 and s_2 from other co-signers,
16. Compute $s = \sum_{j=1}^{3} s_j \bmod n$,
17. The signature is $\sigma = (R, s)$.

EC-Schnorr Multi-Signature (EC-MuSig, or simply MuSig) – Signing m by each user i in $\{1,2,\ldots,u\}$ –

1

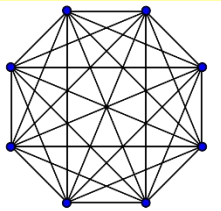
1. Pick k_i at random from $[2, n-2]$,
2. Compute $r_i = k_i G$,
3. Compute $w_i = H_{com}(r_i)$,
4. Broadcast w_i to all other co-signers,
5. Receive w_j from all other co-signers,

2

6. Broadcast r_i to all other co-signers,
7. Receive r_j from all other co-signers,
8. Proceed only if $w_j = H_{com}(r_j)$, for all $j = 1, 2, \ldots, n$,

3

9. Compute $a_j = H_{agg}(L, Q_j)$, for all $j = 1, 2, \ldots, u$,
10. Compute the aggregate public key $Q_{agg} = \sum_{j=1}^{u} a_j Q_j$,
11. Compute $R = \sum_{j=1}^{u} r_j$,
12. Compute $c = H_{sig}(Q_{agg}, R, m)$,
13. Compute $s_i = k_i + c a_i d_i \bmod n$,
14. Broadcast s_i to all other co-signers,
15. Receive s_j from all other co-signers,
16. Compute $s = \sum_{j=1}^{u} s_j \bmod n$,
17. The signature is $\sigma = (R, s)$.



$$\sigma = (R, s)$$

- **Note that**
  - $R = \sum_{j=1}^{u} r_j = \sum_{j=1}^{u} k_j G = \left(\sum_{j=1}^{u} k_j\right) G$
  - $s = \sum_{j=1}^{u} s_j \bmod n = \sum_{j=1}^{u} (k_j + c a_j d_j) \bmod n$

$$= \sum_{j=1}^{u} k_j + c \sum_{j=1}^{u} (a_j d_j) \bmod n$$

- **Further note the aggregate public key Q_{agg}:**
  - $Q_{agg} = \sum_{j=1}^{u} a_j Q_j = [\sum_{j=1}^{u} (a_j d_j)] G$,
    where $a_j = H_{agg}(L, Q_j)$.

EC-Schnorr Multi-Signature – verification

- To verify (m, R, s),
  - Get the ordered list of all public keys:
    $L = \{Q_1, Q_2, \ldots, Q_u\}$
  - Compute the aggregate public key
    $Q_{agg} = \sum_{j=1}^{u} a_j Q_j$, where $a_j = H_{agg}(L, Q_j)$,
  - Compute $c = H_{sig}(Q_{agg}, R, m)$,
  - Accept the signature only if

$$R = sG - cQ_{agg}$$

Indistinguishability

	Single party signature	EC-MuSig	Note
Signature	(r, s)	(R, s)	Same size
Public key	Q (for user i)	$Q_{agg} = \sum_{j=1}^{u} a_j Q_j$ (aggregate)	Same size
Verification	**Check if** $r = sG - \mathrm{Hash}(r, Q_i, m)Q$	**Check if** $R = sG - H_{sig}(Q_{agg}, R, m)Q_{agg}$	Identical (choose $H_{sig} = Hash$)

EC-Schnorr (t, u) Threshold Signature

- **Goal**
  - **At least t out of u users are required to collectively create a valid signature**
  - **Signature is compact: same size as a single Schnorr signature**
  - **Can be verified by anyone**
- **Let $T \in \{1, 2, \dots, t\}$ be a subset of t users.**
- **Let $L = \{Q_1, Q_2, \dots, Q_u\}$ be the ordered set of all u users' public keys**
- **Let L_T be the ordered set of public keys of all t users' in T.**

EC-Schnorr (t, u) Threshold Signature
– Signing m by each user i in T –

1

1. Pick k_i at random from $[2, n-2]$,
2. Compute $r_i = k_i G$,
3. Compute $w_i = H_{com}(r_i)$,
4. Broadcast w_i to all other co-signers in T,
5. Receive w_j from all other co-signers in T,

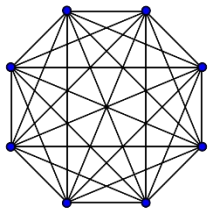
2

6. Broadcast r_i to all other co-signers in T,
7. Receive r_j from all other co-signers in T,
8. Proceed only if $w_j = H_{com}(r_j)$, for all $j \in T$,

3

9. Compute $a_j = H_{agg}(L_T, Q_i)$, for all $j \in T$,
10. Compute $Q_{agg} = \sum_{j \in T}(a_j Q_j)$,
11. Compute $R = \sum_{j \in T} r_j$,
12. Compute $c = H_{sig}(Q_{agg}, R, m)$,
13. Compute $s_i = k_i + c a_i d_i \bmod n$,
14. Broadcast s_i to all other co-signers in T,
15. Receive s_j from all other co-signers in T,
16. Compute $s = \sum_{j \in T} s_j \bmod n$,
17. The signature is $\sigma = (R, s)$.

$u - t$ users unavailable or sitting out

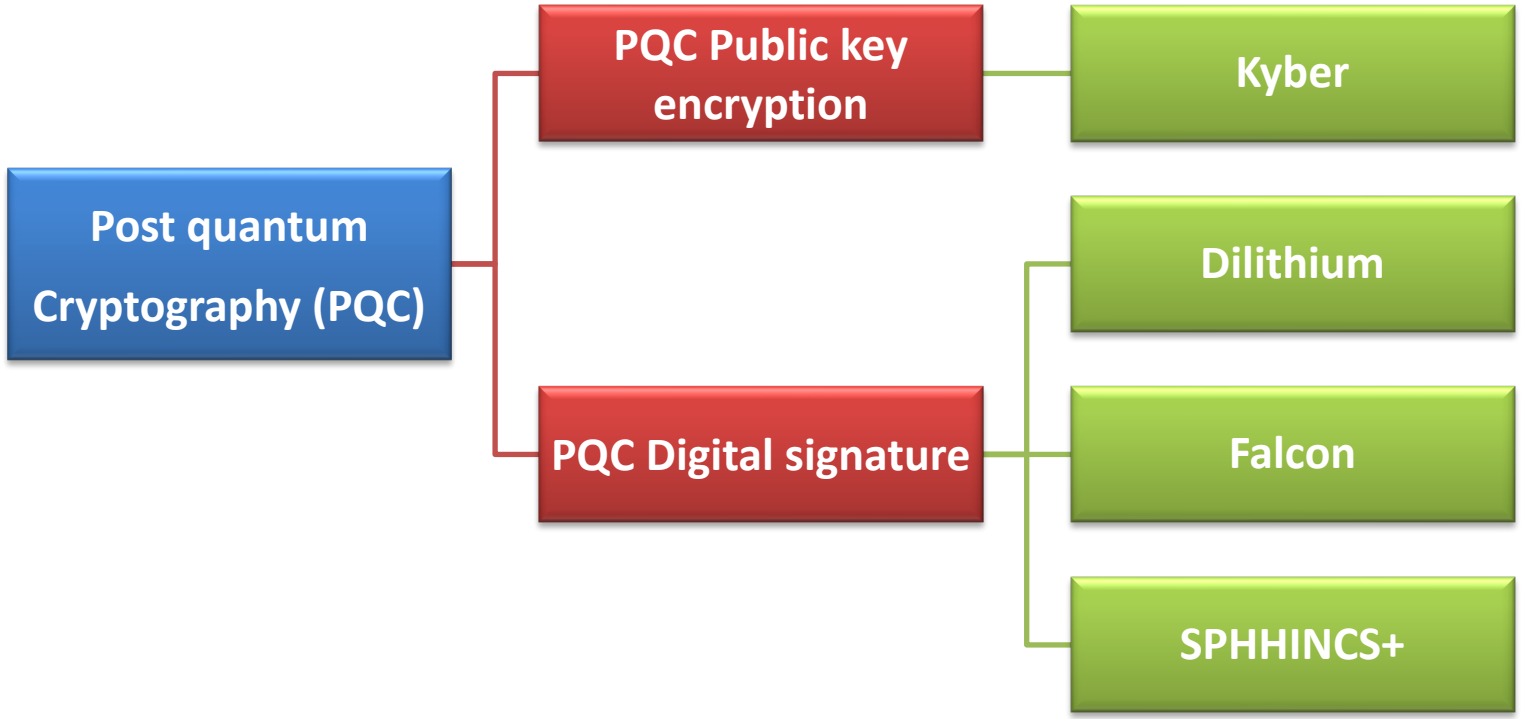


Schnorr (t, u) Threshold Signature – Verification –

- To verify (m, R, s) for a t-subset T
  - Get the ordered list of all public keys $L = \{Q_1, Q_2, \dots, Q_u\}$
  - From L construct L_T, the ordered set of public keys of all t users in T.
    - If L_T, instead of T, is given, make sure $L_T \subseteq L$; Derive T from L_T.
  - Compute the aggregate public key $Q_{agg} = \sum_{j \in T}(a_j Q_j)$, where $a_j = H_{agg}(L_T, Q_i)$,
  - Compute $c = H_{sig}(Q_{agg}, R, m)$,
  - Accept the signature only if $R = sG - cQ_{agg}$.

Post-Quantum Cryptography (PQC)

- **Required to build new blockchains that can withstand attacks that harness the enormous power of future quantum computers**



Reference for Digital Signature

1. FIPS PUB 186-4, *Digital Signature Standard* (DSS), 2013, available at https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

2. G. Maxwell, A. Poelstra, Y. Seurin and P. Wuille: *Simple Schnorr Multi-Signature with Applications to Bitcoin*, 2018. https://eprint.iacr.org/2018/068

3. CS 645/745 “Modern Cryptography” is offered in Spring semester
    1. Foundation of cryptography
    2. Important cryptographic algorithms and protocols
    3. Security against emerging quantum computer
    4. Many more topics