# Complexity, Cryptography, and Financial Technologies

## Lecture 8 – Digital Signature

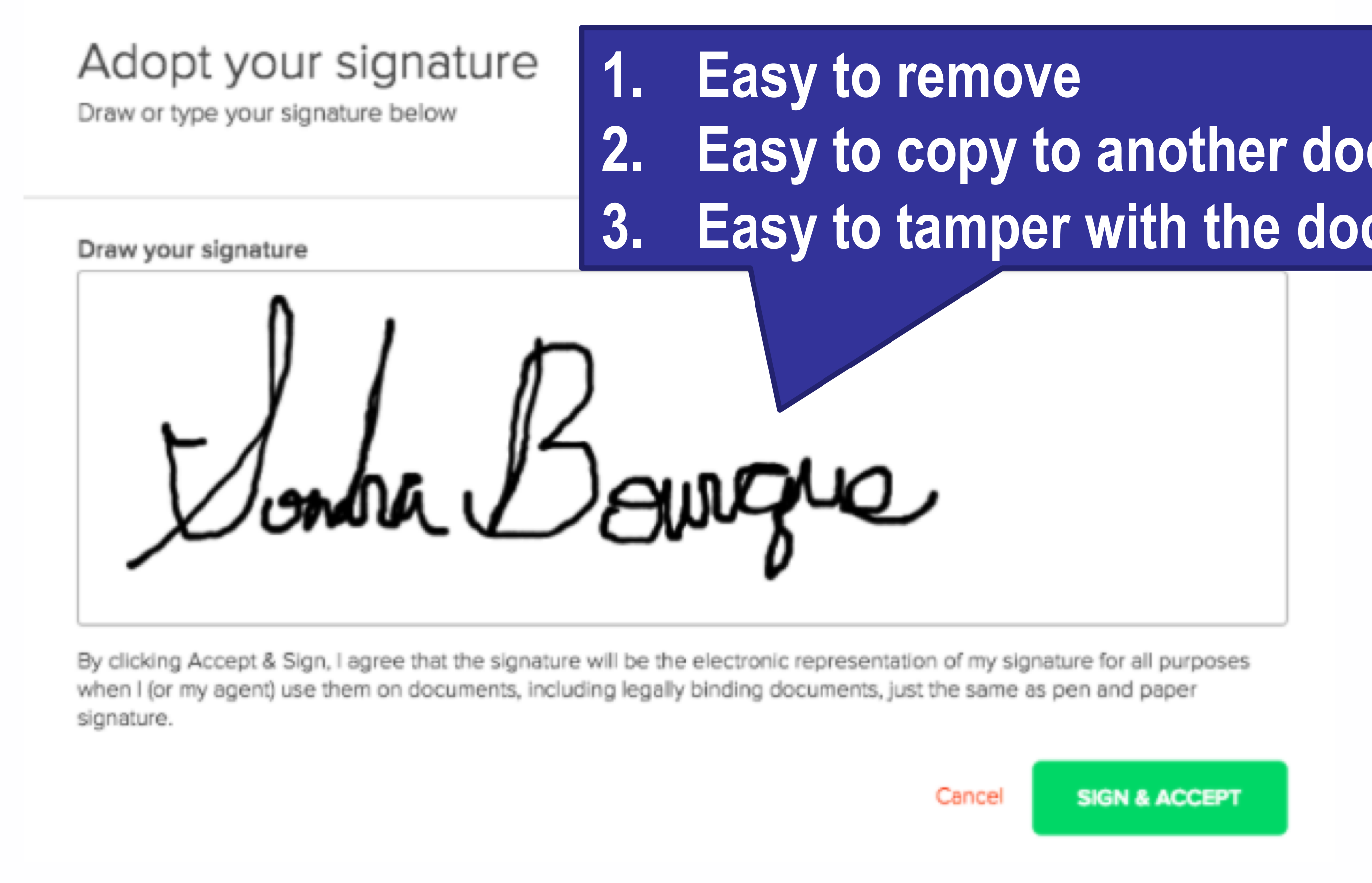
Chan Nam Ngo

# Signature in digital form?


**Source: https://www.proposify.com/blog/how-to-use-electronic-signatures**

# Signature in digital form? (2)

1. Easy to remove
2. Easy to copy to another document
3. Easy to tamper with the document




Source: https://www.proposify.com/blog/how-to-use-electronic-signatures

# Signature in digital form? (3)


1. Easy to remove
2. Easy to copy to another document
3. Easy to tamper with the document

Digital signatures must be:

1. Bound to the signer
2. Bound to the signed message
3. Easy to be verified

Source: https://www.proposify.com/blog/how-to-use-electronic-signatures

# Digital Signature - ECDSA (NIST P-256, SHA-256)

- **Verifying Key**
  - 0402bce88f7016f91de2a196d95c7bff83479efe025f8dfd31f1122cb1a5df1bf9019c524236c438b50e88ce77422b4a4c15c08bed702fea32c9502824d534a2b0
- **Signing Key**
  - 117ba01c2c5140767d2a916e30780ad0d4e538a5b709f98ab1c5dc923647e60f
- **Message**
  - Hello World!
- **Digital Signature**
  - 3046022100ff41e1cee95fbd3b02f78977307d325f83fd112c6911564fd87ba7fd4bb4114802210Ofced62e118ebd3684bd1fc6612075ede28f88dccbcd02a858a2bc21d996db4df

**Generated with: https://kjur.github.io/jsrsasign/sample/sample-ecdsa.html**

# Digital Signature

- **Key Generation**
  - (vk, sk) ← KeyGen()
    - vk is called the (public) verifying key
    - sk is called the (private) signing key
- **Signature Generation**
  - s ← Sign(sk,m)
    - m is the (PUBLIC) message to be signed
    - s is the signature on m
- **Signature Verification**
  - {0,1} ← Verify(vk,s,m)
    - Return 1 if s is a valid signature on m
    - Return 0 otherwise

# Digital Signature Algorithm (DSA)

UNIVERSITY OF TRENTO

- **(vk,sk) ← KeyGen()**
  - Fix a large prime p, a group $Z_p$ and a generator g
  - Randomly pick x in $Z_p$
  - Compute $y = g^x$
  - Return vk = (p,g,y) and sk = (x)
- **s ← Sign(sk,m)**
  - Randomly pick k in $Z_p$ s.t. gcd(k,p-1) = 1
  - Compute $R = g^k \pmod{p}$
  - Compute $S = (m-xR)/k \pmod{p-1} = (m - xg^k)/k$ → $m = Sk + xR$
  - Return s = (R,S)
- **{0,1} ← Verify(vk,s,m)**
  - Return 1 if $g^m = y^R R^S \pmod{p-1}$
    - $g^m = g^{Sk + xR} = g^{xR} g^{kS} = y^R R^S$

# EC Digital Signature Algorithm (ECDSA)

- **(vk,sk) ← KeyGen()**
  - Fix a large prime p,
  - Generate an EC

    $x, y \in Z_p$, $y^2 = x^3 + bx + c$
  - Fix a based point G on E
  - Randomly pick k in $Z_p$
  - Compute Y = kG
  - Return vk = (p,G,Y) and sk = (k)
- **s ← Sign(sk,m) (m is not a point but an integer)**
  - Randomly pick r in $Z_p$ s.t. gcd(r,n) = 1 (n is the number of points in E)
  - Compute R = rG = (u,v)
  - Compute S = (m - ku)/r (mod n) → m = Sr + ku
  - Return s = (R,S)
- **{0,1} ← Verify(vk,s,m)**
  - Return 1 if mG = uY + SR
    - mG = (Sr + ku)G = ukG + SrG = uY SR

# RSA Signature

UNIVERSITY OF TRENTO

- **$\phi$(N) – Euler's totient function**
  - numbers of positive integers in [1,N] that coprime N
  - e.g. $N = 10$, $\phi(10) = 4$ (1,3,7,9 coprime 10)
- **(vk,sk) ← KeyGen()**
  - Fix two large primes p and q
  - Compute $N = p*q$
  - Pick e s.t. $1 < e < \phi(N)$ and $\gcd(e, \phi(N)) = 1$
  - Find d s.t. $e*d = 1$
  - Return vk = (N,e) and sk = (d)
- **s ← Sign(sk,m)**
  - Return $s = m^d$
- **{0,1} ← Verify(vk,s,m)**
  - Return 1 if $m = s^e$
    - $s^e = m^{de} = m^1 = m$

- **Blind Signature**
  - A client can obtain a signature from a server for a message m without the server knowing m.
- **5 algorithms**
  - Key Generation
    - (vk, sk) ← BKeyGen()
      - vk is called the verifying key
      - sk is called the signing key
  - Message Blinding
    - (x,r) ← Blind(vk,m)
      - m is the message to be signed
      - r is called the blinding factor
      - x is called the blinded message
  - Blind Signing
    - y ← Sign(sk,x)
      - y is called the blind signature
  - Signature Unblinding
    - s = Unblind(y,r)
      - s is the signature on m
  - Signature Verification
    - {0,1} ← Verify(vk,m,s)
      - Return 1 if s is a valid signature on m
      - Return 0 otherwise

# RSA Blind Signature

- **(vk,sk) ← KeyGen()**
  - Fix two large primes p and q
  - Compute $N = p*q$
  - Fix e s.t. $1 < e < \phi(N)$ and $\gcd(e, \phi(N)) = 1$
  - Find d s.t. $e*d = 1$
  - Return vk = (N,e) and sk = (d)
- **(x,r) ← Blind(vk,m)**
  - Randomly pick r s.t. $\gcd(r,N) = 1$
  - Compute $x = r^e m$
- **y ← Sign(sk,m)**
  - Return $y = x^d = (r^e m)^d = r^{ed} m^d = r^1 m^d = rm^d$

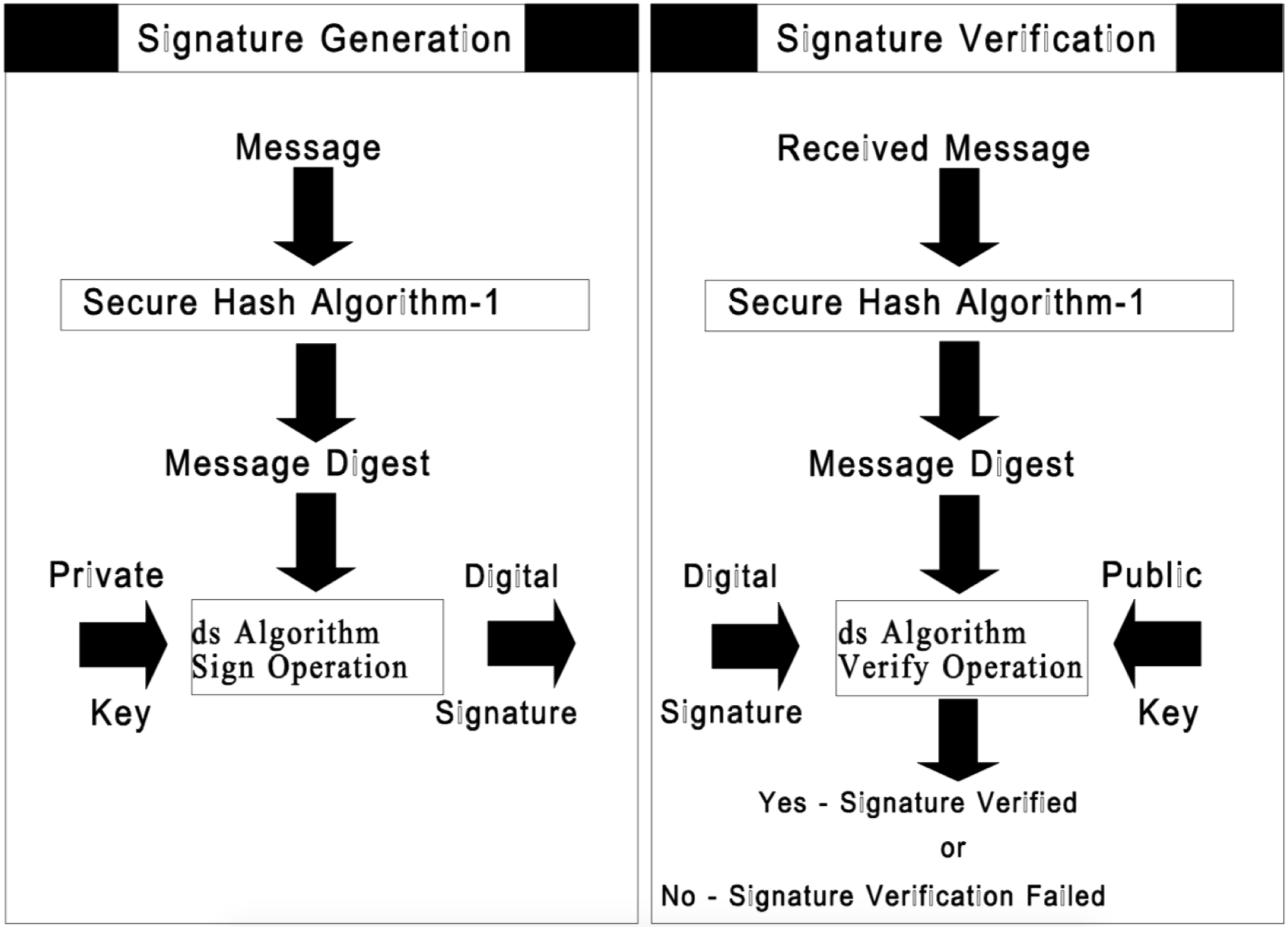
# RSA Blind Signature (2)

- **$s \leftarrow$ Sign(sk,m)**
  - Return $s = x^d = (r^e m)^d = r^{ed} m^d = r^1 m^d = rm^d$
- **$s$ = Unblind(y,r)**
  - Return $s = y/r = rm^d/r = m^d$
- **$\{0,1\} \leftarrow$ Verify(vk,s,m)**
  - Return 1 if $m = s^e$
    - $s^e = m^{de} = m^1 = m$

# Hashing and Signing

- **DS schemes only support short messages, e.g.**
  - DSA, ECDSA → m in $Z_p$ (256 bits)
  - RSA → m in $Z_n$ (2048 bits)
- **But messages are usually very long**
  - contracts, transactions, etc. can be some KBs to MBs
    - could be a PDF file
- **Hash then sign strategy**
  - Fix a hash function H, obtain the message digest m = H(M)
  - We sign only the message "digest"
    - s = Sign(sk,m)
    - M can be a big document, e.g a PDF
    - m is a short message digest obtained via hashing
  - Verification for a signature s of a document M is done in two steps
    - m =? H(M)
    - {0,1} ← Verify(vk,s,m)

# Digital Signature Standard – FIPS 186-2



# Suggested Reading



- **Handbook of Applied Cryptography, book by Menezes, C. van Oorschot and Vanstone**
  - See Chapter 11 for Digital Signatures
  - Also available on the author’s website
- **Introduction to Cryptography with Coding Theory, book by Trappe and Washington.**
  - Chapter 9 is on Digital Signatures
- **NIST standard FIPS 186-2**
  - https://csrc.nist.gov/csrc/media/publications/fips/186/2/archive/2000-01-27/documents/fips186-2.pdf
  - see main content (Sections 4, 5 and 6) for DSA