Intermediate Protocols, and digital cash applications

Hash applications: digital cash protocols

- Financial transactions online are typically tied to a customer's personal information
  - Necessary to verify that a customer has money, and to commit the transfer of money to vendor.
- In reality, people can use cash to buy goods without revealing any identifying information
- Can such a concept exist online?

Attempt 1

- Bob founds a bank to issue digital $50 bills:
  - Bob charges you $51 ($1 service fee)
  - Bob sends you the message:

M = {I, Bob, hereby agree to pay US$50 to whoever presents me with this message. Use the attached signature with my public key to verify that it is legitimately issued.},
H = [hash M],
S = [decrypt D_B H]

Attempt 2

- Alice (anonymously) mails Carol the digital money, Carol can verify its value, accepts it as payment
- What if Carol tries to cash it twice, or Alice tries to spend it twice?

M = {I, Bob, hereby agree to pay US$50 to **the first person who presents me with this message**. Use the attached signature with my public key to verify that it is legitimately issued.},
H = [hash M],
S = [decrypt D_B H]

Attempt 3

- Both Bob and Carol need a way to determine if the digital banknote has already been spent.
- Solve with a serial number, and a publicly accessible database of previously spent money:

M = {I, Bob, hereby agree to pay US$50 to **the first person who presents me with this message**. Use the attached signature with my public key to verify that it is legitimately issued.
SERIAL NUMBER: [huge random nonce]},
H = [hash M],
S = [decrypt D_B H]

Problem

- Alice can buy and use a digital bank note, Carol and Bob can verify that it is unspent. But Alice's transactions are now traceable!
- Bob records serial number when selling Alice a banknote, and when a vendor cashes it, linking Alice to Carol. Making the money secure destroys anonymity.

M = {I, Bob, hereby agree to pay US$50 to **the first person who presents me with this message**. Use the attached signature with my public key to verify that it is legitimately issued.
SERIAL NUMBER: [huge *traceable* nonce]},
H = [hash M],
S = [decrypt D_B H]

Attempt 5

- Alice fills out the banknote message M, commits to it with a hash, and Bob signs H without seeing M.
- Technically possible, but dumb: Bob has no idea what he is signing.

Alice: M = {I, Bob, hereby agree to pay US$50 to the first person who presents me with this message. SERIAL#: [huge random nonce]}}
Alice→Bob: H = hash[M],
Bob→Alice: H = [hash M], S = [decrypt D_B H]
Alice→Carol: (M, H, S) ← alleged $50 bill

Solution: cut and choose protocols

How can you trust a contract you aren't allowed to read?

1. Alice generates 1,000 separate $50 banknote messages $\{M_1, M_2, .. M_{1000}\}$ each with a different random nonce but otherwise identical content.

2. She computes H_1=hash[M_1]…H_{1000}=hash[M_{1000}], and sends the hashes $\{H_1 \ldots H_{1000}\}$ to Bob

3. Bob chooses one hash at random, informing Alice of his choice.

4. Alice sends the 999 messages corresponding to the hashes Bob did *not* choose opening 999 commitments.

Solution: cut and choose protocols

How can you trust a contract you aren't allowed to read?

5. Bob computes the hashes of these messages, verifying that they are all valid and identically formatted $50 banknotes.

6. Bob reasonably concludes that the unopened commitment is also a valid $50 banknote, and signs its hash without seeing the message.

7. Alice pays $51 and uses her valid $50 banknote with a unique serial number Bob does not know.

Problems remaining

- Alice could still double-spend a file; Carol must not only verify the money is unspent, but cash it immediately.
- Even immediate cashing doesn't prevent Alice from making two payments simultaneously.
- Like real money, digital bills have fixed denominations. You can't spend part of the $50, but can only receive "change."

Bigger problem: Bob can still track a money order using its hash

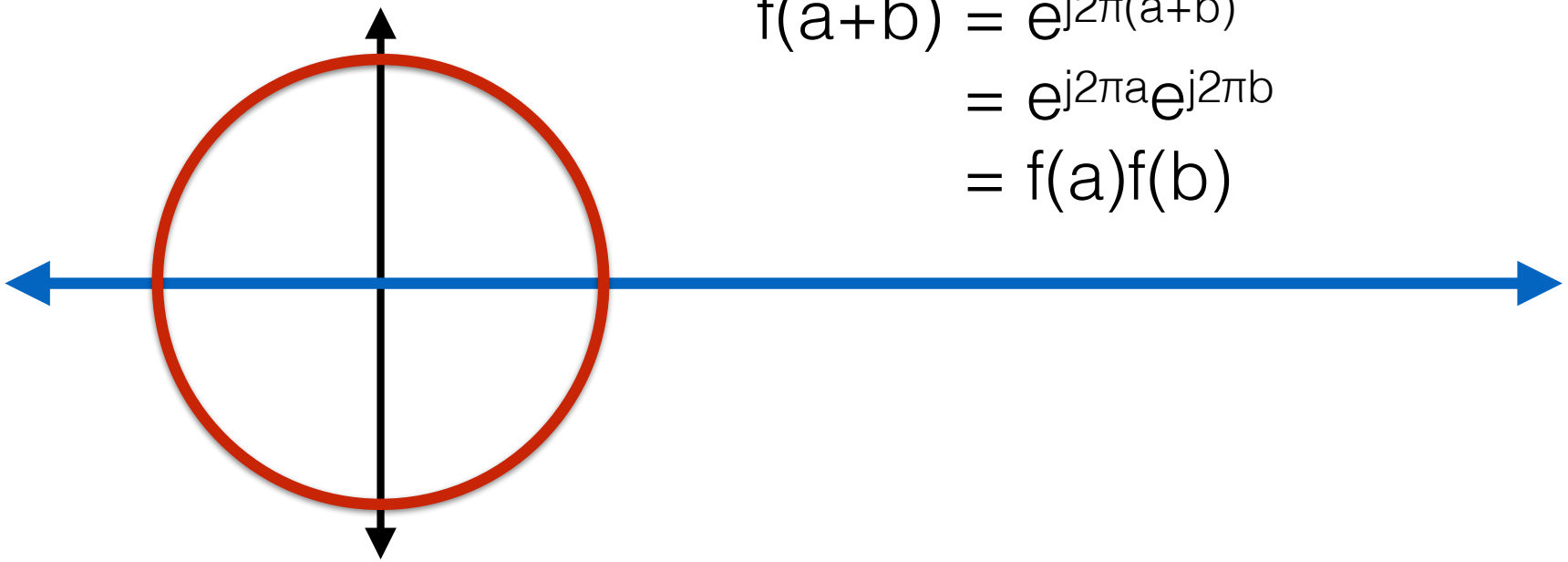
- Bob signs H_K=hash[M_K], does not see serial number in M_K, but provides signature.
- But wait, if M_K is eventually redeemed, can't Bob hash it, and see the hash he signed for Alice?
- This still allows Bob to connect the banknote he sold to Alice with the banknote cashed by Carol.

Homomorphism

Consider a function $f(t) = e^{j2\pi t}$

This maps a real number t to a complex number on the unit circle of the complex plane

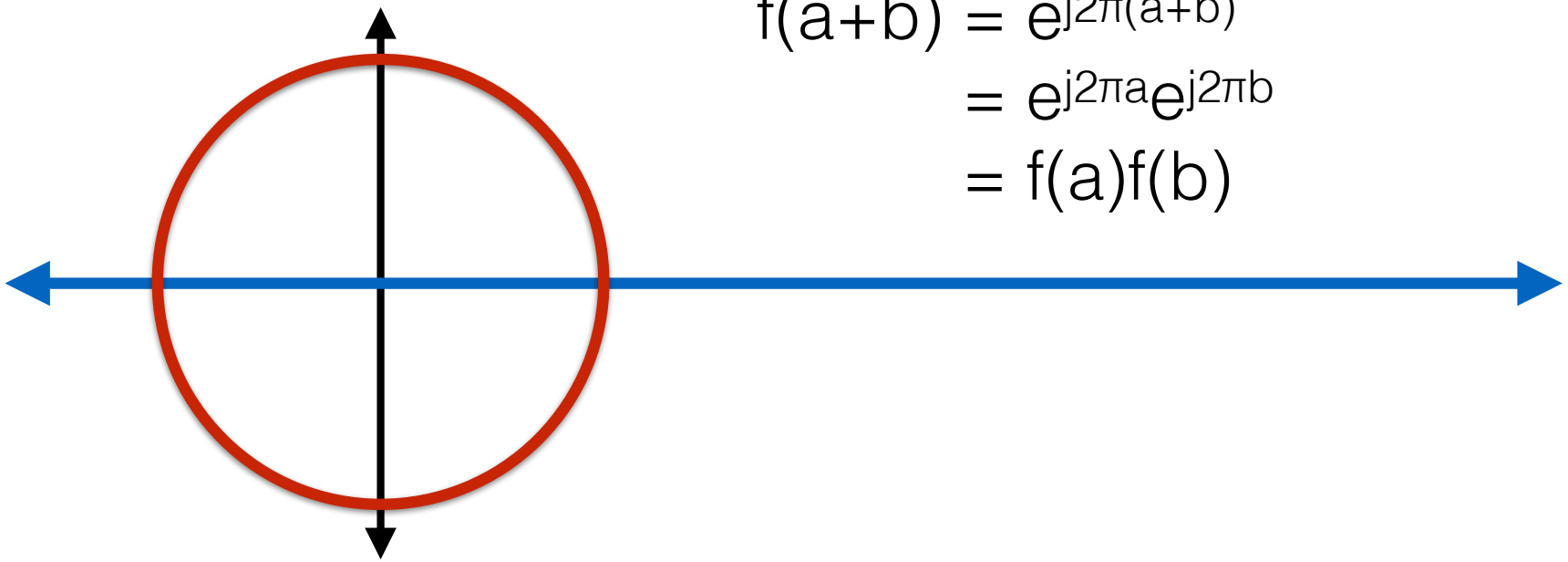
$$f(a+b) = e^{j2\pi(a+b)}$$
$$= e^{j2\pi a} e^{j2\pi b}$$
$$= f(a)f(b)$$



Homomorphism

A homomorphism is a mapping (function) from one set of things to another, that preserves arithmetic in some way

We see the pattern $f(a \boxplus b) = f(a) \bigstar f(b)$, for some operations $\boxplus$ and $\bigstar$.



$$f(a+b) = e^{j2\pi(a+b)}$$
$$= e^{j2\pi a} e^{j2\pi b}$$
$$= f(a)f(b)$$

Homomorphism examples

Set S	operation	function t=f(s)	Set T	operation
Real numbers	+	$f(t)=e^{jt}$	Unit circle in complex plane	×
Integers	+	f(n) = n mod 2 (1 if n is odd)	Bit values {0,1}	XOR
Sets	Union, Intersection, Difference	$f(A)=\sum 2^n$ for all whole numbers {0,1,2..} in A	Whole numbers	OR AND XOR
Square matrices	Matrix multiplication	f(A) = det(A)	Real numbers	×

Homomorphism in encryption

- In RSA (public-key) encryption, we have the property
  $Encrypt_K[A \times B] = Encrypt_K[A] \times Encrypt_K[B]$
  $Decrypt_K[A \times B] = Decrypt_K[A] \times Decrypt_K[B]$
  (all arithmetic is modulo some large number n)

- Consider the protocol:

  1. A: N = [nonce], Y = Encrypt[N]

  2. A→B: Z = M×Y (M==message we want signed)

  3. B→A: S = Decrypt[M×Y] = Decrypt[M]×N

Homomorphism in encryption

- Consider the protocol:

  1. A: N = [nonce], Y = Encrypt[N]

  2. A→B: Z = M×Y (M==message we want signed)

  3. B→A: S = Decrypt[M×Y]
     = Decrypt[M]×Decrypt[Y]
     = Decrypt[M]×N

  4. A: $S \times N^{-1}$ = signature of message M.

- Bob does not see the message being signed

Blinding

- A “blinding protocol” is a type of cryptographic protocol that conceals a piece of information
- Often involves deliberate obfuscation by a “blinding factor”
- In some protocols, it is helpful for the blinding operation to commute with some other primitive

```
encrypt[ blind[X] ] = blind[ encrypt[X] ]
```

In other words, we need homomorphic primitives.

What can we do with blinding?

1. Alice generates 1,000 separate $50 banknote messages $\{M_1, M_2, .. M_{1000}\}$ each with a different random nonce but otherwise identical content.

2. She computes H_k=hash[M_k] and B_k=blind[H_k], sending $\{B_1, B_2, .. B_{1000}\}$

3. Bob chooses one hash at random, informing Alice of his choice.

4. Alice sends the 999 messages and unblinds those 999 hash values.

What can we do with blinding?

4. Bob chooses one hash at random, informing Alice of his choice.

5. Alice sends the 999 messages and unblinds those 999 hash values.

6. Bob signs the remaining B_k=blind[H_k], with

   S = sign[B_k]=sign[blind[H_k]] = blind[sign[H_k]]

7. Alice unblinds S to get sign[H_k].

How do you “unblind” a message?

- Message M, random number R, blinding factor T=Encrypt[R]
  Blinded message is C=(MT)

- When asked to “unblind” C, Alice reveals T, so Bob can compute $M=CT^{-1}$

- But wait a minute, so what? Does that prove anything at all? That just proves that C is some message times a thing.

How do you “unblind” a message?

- Cheater Alice:
  Evil message M*, innocent message M
  random number R, blinding factor T=Encrypt[R]
  Blinded evil message is C=(M*T)

- C also equals MK for some $K=CM^{-1}$

- Alice sends K to “prove” that C is a blinded version of innocent message M; it is actually a blinded version of evil message M*

How do you “unblind” a message?

1. Better Alice:
   Message M
   Nonce N_A
   Hash value H = hash[Alice,N_A]
   Pseudo-random number R = PRNG[H],
   blinding factor T=Encrypt[R]
   Blinded message is C=(MT)

2. When challenged to “unblind,” Alice reveals N_A

3. Bob can compute T, $M=CT^{-1}$,

   How does this fix the problem?

How do you “unblind” a message?

1. Better Cheater Alice:
   Message M, evil message M*
   Nonce N_A
   Hash value H = hash[Alice,N_A]
   Pseudo-random number R = PRNG[H],
   blinding factor T=Encrypt[R]
   Blinded message is C=(M*T)

2. Cheater Alice could compute K with C=(MK), but K is not the encryption of the hash of something she knows

   Alice can not:
   Find R with K=Encrypt[R] (she can’t break encryption)
   Find seed S with R=PRNG[S] (she can’t break PRNG)
   Find input N with S = hash[Alice,N] (she can’t break hash)

Fixing the double-spending problem

- Problem one: Carol could make a copy of the money when verifying it, could refuse to accept it, but cash it anyway.

- Solution: make serial number a hash

M = {I, Bob, hereby agree to pay US$50 to the first person who presents me with this message, **and the nonce that hashes to its serial number. SERIAL NUMBER: [hash N]**},
N = [nonce] (separate from M)
S = [sign M] = [decrypt D_B M]

Fixing the double-spending problem

1. Alice gives (S,M) to Carol, Carol can check the money is legitimate, and not yet spent

2. Carol not allowed to cash it without N; Alice hands over N if money is accepted.

M = {I, Bob, hereby agree to pay US$50 to the first person who presents me with this message, **and the nonce that hashes to its serial number. SERIAL NUMBER: [hash N]**},
N = [nonce] (separate from M)
S = [sign M] = [decrypt D_B M]

Fixing the double-spending problem

- Problem 2: Carol needs to cash money immediately to be sure Alice isn't spending the money order somewhere else
- Can we simply punish Alice if she double spends, i.e. call the cops?
- NO: firstly, Alice may be engaging with Carol anonymously. Secondly, we can't tell whether Alice has double-spent the money, or if a vendor is double-cashing the money, and blaming Alice.

Fixing the double-spending problem

1. Alice has bank account with Bob, with account number A that identifies her

2. Let N = Encrypt[A, [nonce]] with E_B

3. Split string N into two parts, N = N_1N_2

4. New money order has two serial numbers, hash[N_1] and hash[N_2]

Fixing the double-spending problem

M = {I, Bob, hereby agree to pay US$50 to the first person who presents me with this message, **and the nonce that hashes to one of these serial numbers.**
SERIAL NUMBER ONE: [hash N_1]
SERIAL NUMBER ONE: [hash N_2]
},
N = N_1N_2 = [encrypt E_B A [nonce]]
S = [sign M] = [decrypt D_B M]

Fixing the double-spending problem

- When Alice unblinds one of these messages, she hands over N. Bob verifies that this hashes to the serial numbers, and decrypts to Alice's account #

M = {I, Bob, hereby agree to pay US$50 to the first person who presents me with this message, **and the nonce that hashes to one of these serial numbers.**
SERIAL NUMBER ONE: [hash N_1]
SERIAL NUMBER ONE: [hash N_2]
},
N = N_1N_2 = [encrypt E_B A [nonce]]
S = [sign M] = [decrypt D_B M]

Fixing the double-spending problem

1. Alice presents (M,S) to Carol.

2. Carol verifies signature, will accept money
   (NO CHECKING A DATABASE FOR DOUBLE SPENDING)

3. Carol flips a coin, and demands either N_1 or N_2

4. Alice reveals one of these nonce halves, “unlocking” the money.

5. Carol can verify the nonce half hashes to a serial number on the money, and can now cash it.

Fixing the double-spending problem

1. Carol cashes the money, and Bob pays $50

2. Given either N_1 or N_2, Bob doesn't have enough information to determine identity of spender.

3. If Alice multiple-spends the money, eventually she will be challenged to hand over N_1 and N_2 in different transactions, which will be given to Bob

4. If Bob collects them all, can compute Alice's identity as Decrypt[N_1N_2], call the cops

Fixing the double-spending problem

1. This still allows some double-spending with some probability, but this is easily amended.

2. Let N = Encrypt[E_B, A, [nonce]]

3. Split string N into m parts, N = $N_1N_2\ldots N_m$

   To “unlock” a money order, Alice now has to hand over all but one of the nonce parts. Increases probability that double spending will betray Alice’s account number.

Fixing the double-spending problem

1. Carol no longer needs to consult a lookup table, or cash money right away — it's an offline protocol!

2. Bob can have a policy of redeeming any double-spent money order as long as the serial numbers implicate Alice (prevents vendor from double-cashing!)

3. Alice can conduct a financial transaction anonymously, but be identified in the specific case that she attempts fraud.

One final weird problem

- If Alice is anonymous when transacting with Carol, can't she just cheat by quitting the protocol?
- Alice sends Carol and Dave (M,S), Carol and Dave demand all but one nonce piece
- If they ask for the same set, Alice wins, hands them over, and has double-spent the money
- If they ask for different sets, Alice just logs off, waits a random amount of time, and tries again.

Better solution: oblivious transfer

- I have separate pieces of information $\{M_1, M_0\}$
- You want one of them, but you don't want me to know which one I am giving you.
- Imagine if a money order has two nonce halves, and Carol can demand one without Alice knowing which was transmitted. Then she can't game the system with Carol and Dave.

Blinded transfer of message M

1. Carol→Alice blinding factor $T=\text{Encrypt}[E_A, R]$
2. Alice→Carol: $C=M\times\text{Decrypt}[D_A, T] = M\times R$
3. Carol: $M = CR^{-1}$

   This only works because Carol knows the decryption of T, and she only knows that because she made T the encryption of a known value R.

Blinded transfer of message M

1. Carol: blinding factor T_1=Encrypt[E_A, R],
   T_0 = [string reverse T_1]

2. Carol→Alice: {T_1, T_0} (unlabeled, in random order, no way to tell which message is the forwards one)

3. Alice→Carol:
   C_0=M_0×Decrypt[D_A, T_0] = M_0×R
   C_1=M_1×Decrypt[D_A, T_1] = M_1×??????

   Carol only knows the decryption of one blinding factor, and can only unblind one message. In practice, Alice doesn't know which one is reversed, and doesn't know which transmitted message can be unblinded.