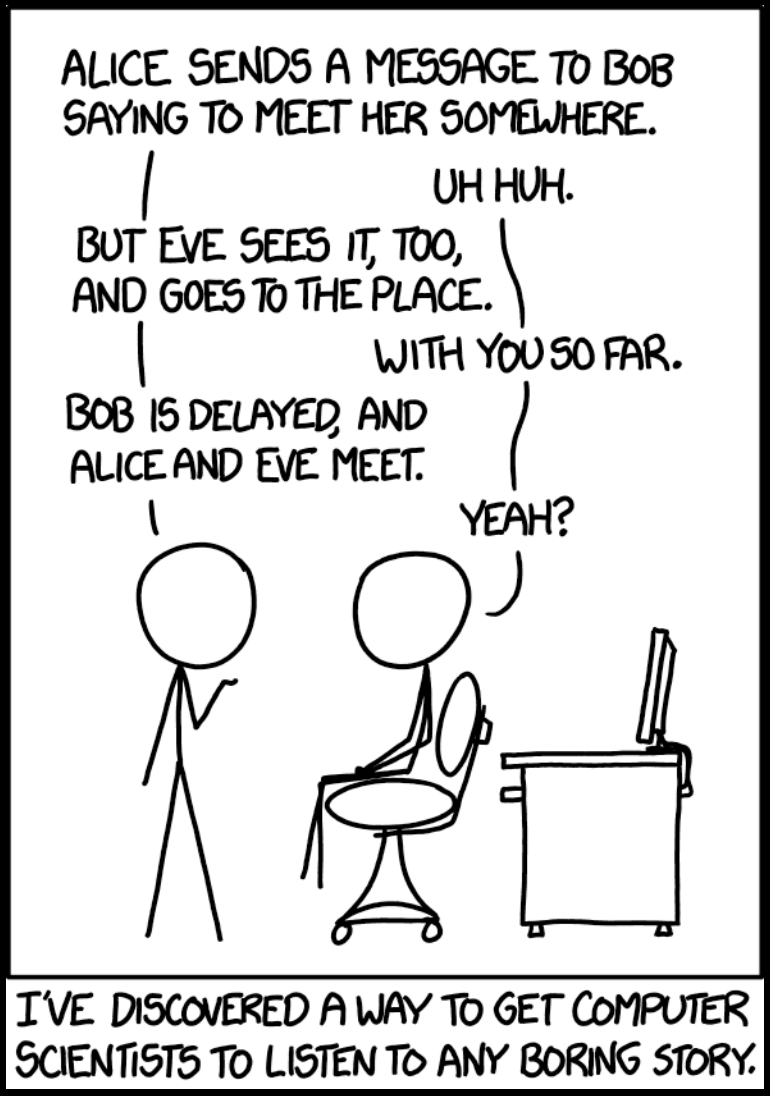

https://xkcd.com/1323/

Cryptocurrencies & Security on the Blockchain

Digicash, Part 1: Blinded Signatures

Prof. Tom Austin
San José State University

Lab 3 Review

DigiCash

- Created by David Chaum
- A *centralized* cryptocurrency
  - Relies on *trusted third party* (TTP)
- Anonymous transactions
- Uses *blind signatures*

Review: Digital Signatures

- Uses two separate keys
  - Public key is known by everyone
  - Private key known only to the owner
- Encryption
  - Public key encrypts
  - Private key decrypts
- Signing
  - Private key encrypts (signs)
  - Public key decrypts (verifies)

Public key is (N,e)

Private key is d

To encrypt message M:

$$C = M^e \bmod N$$

To decrypt ciphertext C:

$$M = C^d \bmod N$$

Public key is (N,e)

Private key is d

To sign message M:

$S = M^d \bmod N$

To verify sig. S for message M:

$M' = S^e \bmod N$

and verify that M = M'

Blinded signatures

- Motivation: Sender wants a notary to sign a document without revealing the contents.
- Analogy: Signing a piece of carbon paper through an envelope.
- Can the sender get the notary to sign anything it wants?

Blind Signature Properties

- Signature function and multiplication must be commutative
- Signature remains valid after unblinding
- Signer cannot determine what was signed, until it is unblinded
  - Signer does not even know when when a particular document was signed

Public key is (N,e) **Private key** is d

To sign message M:

$S = M^d \bmod N$

To verify sig. S for message M:

$M' = S^e \bmod N$

and verify that M = M'

RSA Blind Signature process

- User chooses *blinding factor* B
  - B must be cryptographic quality random number
- User calculates:
  $M' = M * B^e \bmod N$
- Bank signs M'
  $S' = M'^d \bmod N$
- User removes blinding factor
  $S = S' / B$
- Note that $S' / B = M^d \bmod N$

Why does it work?

$$M' = M * B^e \bmod N$$

$$S' \equiv M'^d \bmod N$$

$$= (M * B^e)^d \bmod N$$

$$= M^d * (B^e)^d \bmod N$$

$$= M^d * B \bmod N$$

$$S' / B = M^d \bmod N$$

Modular Math

- "Clock math"
- Addition – easy
- Subtraction – easy
- Multiplication – easy
- Division… not so much

Modular Division

To find a / b mod N:

1. Find the *modular multiplicative inverse* of b
   - Denoted b^{-1}
   - Might not be defined
2. Return a * b^{-1} mod N

Modular Multiplicative Inverse

- Multiplicative inverse of b in mod N math
  Find number b^{-1} such that
  - $b * b^{-1} = 1 \bmod N$
- If b-1 exists, then the greatest common divisor of b and N is 1.
  - gcd(b,n) = 1
- Extended Euclidean algorithm calculates b-1
  - Modified version of finding gcd
  - https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

Lab, Part 1: Implement Blind RSA

Download `rsa.js` from the course website. It demonstrates the math for the RSA algorithm.

Add `blind` and `unblind` functions for working with signatures. Test out your solution, then paste these functions into Canvas.

For unblinding, you will need to do modular division. You may find `modularDivision.js` useful.

Blinded Signatures library

- Available through `npm`
  - From your project directory, type
    `npm install blind-signatures`
  - Source code/documentation available at https://github.com/kevinejohn/blind-signatures
- Signatures happen on the *hashes* of the documents

JavaScript tip of the day: destructuring assignment

- Uses pattern matching to break apart more complex values.
- Useful for returning multiple values.
- (Recent versions of JS only).

Returning Multiple Values (old school)

```
function foo() {
  return [1,"one"];
}

let arr = foo();
let n = arr[0];
let s = arr[1];
console.log(n); // Prints '1'
console.log(s); // Prints 'one'
```

Destructuring Assignment with Arrays

```
function foo() {
  return [1,"one"];
}

let [ n, s ] = foo();


console.log(n); // Prints '1'
console.log(s); // Prints 'one'
```

Destructuring Function Parameters

```
function bar({x, y}) {
  return x + y;
}

let o = {x: 3, y: 4};
let z = bar(o);

console.log(z);
```

Back to Blind Signatures Library …

- `blind`: returns
  - Blinded hash of the document
  - Its blinding factor
- `sign`: signs blinded document
- `unblind`: removes blinding factor from signature
- `verify`: determines validity of signature
- `verify2`: same as `verify`
  - uses private key (for efficiency I guess?)
- `messageToHash`: calculates hash of message

Blinding a Document

```
let { blinded, r } =
  blindSignatures.blind({
      message: document,
      N: pubKey.N,
      E: pubKey.E,
  });
```

Signing a Document

```
let signed =
  blindSignatures.sign({
      blinded: blinded,
      key: key,
  });
```

Unblinding a Document

```
let unblinded =
  blindSignatures.unblind({
      signed: signed,
      N: pubKey.N,
      r: r,
  })
```

Unblinding a Document

```
let result =
  blindSignatures.verify({
      unblinded: unblinded,
      N: pubKey.N,
      E: pubKey.E,
      message: document,
});
```

Lab, Part 2

Details in Canvas and on course website.

Starter code is available on the course website.