🃏

Mental Poker

CSYE 7374 — Cryptography and Cybersecurity

By Maharshi Jinandra

The Idea :- http://people.csail.mit.edu/rivest/ShamirRivestAdleman-MentalPoker.pdf

TLDR; The game of **“Mental Poker”** is played just like ordinary poker except that there are no cards: all communications between the players must be accomplished using messages.

The Protocol

Normal Poker	*Mental Poker*
Gather players around the table	Players each run the key generation algorithm, publish their public key and prove that they own the corresponding secret key
Deck of cards	Publicly encode each card value
Shuffle the deck	All players take turns running a shuffle
Give out the cards: top card goes to first player, second card to second player, etc …	Take part in the decryption process to allow players to see their cards: all players but the first partially decrypt the first card, all players but the second partially unmask the second card, etc …
Betting	Betting
Final betting. Open hands and declare a winner	Final betting. Each player can complete the decryption process of their private cards. Compare hands and declare a winner

Commutative Encryption

An encryption algorithm is commutative if

> $$E_{k1}(E_{k2}(m)) = E_{k2}(E_{k1}(m))$$

Also, we can easily figure out that

> $$D_{k1}(D_{k2}(E_{k1}(E_{k2}(m)))) = m$$

We need a public key encryption system for the above protocol to work. With the public key payers can encrypt the cards and with the private keys to decrypt.

First try was a elliptic curve cryptography

Elliptic curve cryptography

This is nothing but a better trapdoor function than RSA.

RSA and Diffie-Hellman were so powerful because they came with rigorous security proofs. The authors proved that breaking the system is equivalent to solving a mathematical problem that is thought to be difficult to solve. That said, factoring is not the hardest problem on a bit for bit basis. Specialized algorithms like the Quadratic Sieve and the General Number Field Sieve were created to tackle the problem of prime factorization and have been moderately successful.

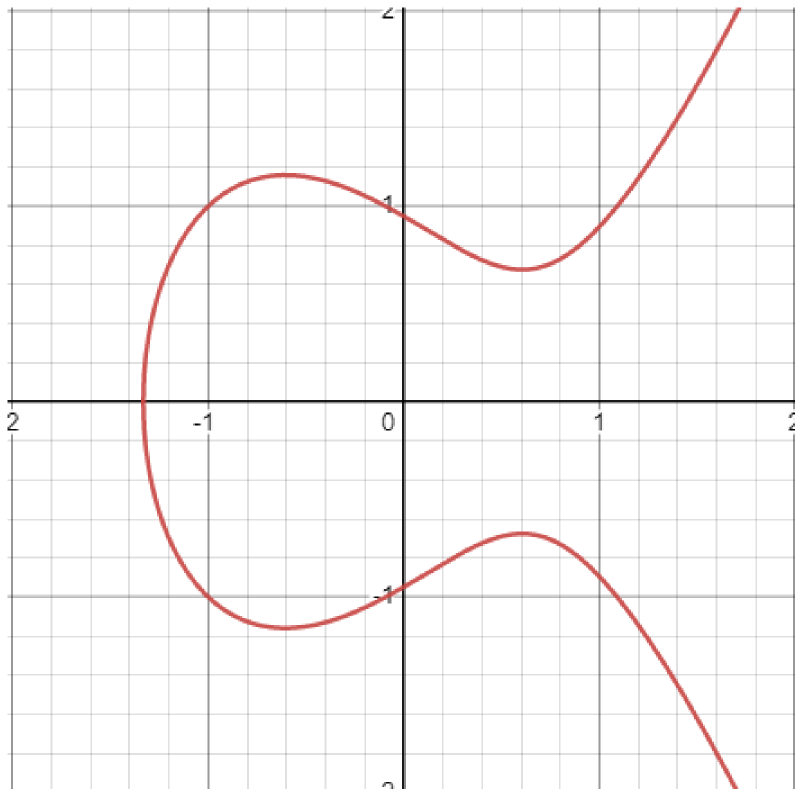
These factoring algorithms get more efficient as the size of the numbers being factored get larger. The gap between the difficulty of factoring large numbers and multiplying large numbers is shrinking as the number (i.e. the key's bit length) gets larger. As the resources available to decrypt numbers increase, the size of the keys need to grow even faster. This is not a sustainable situation for mobile and low-powered devices that have limited computational power. The gap between factoring and multiplying is not sustainable in the long term.

Researchers explored other mathematics-based cryptographic solutions looking for other algorithms beyond factoring that would serve as good Trapdoor Functions. In 1985, cryptographic algorithms were proposed based on branch of mathematics called elliptic curves.

An elliptic curve is the set of points that satisfy a specific mathematical equation. The equation for an elliptic curve looks something like this:

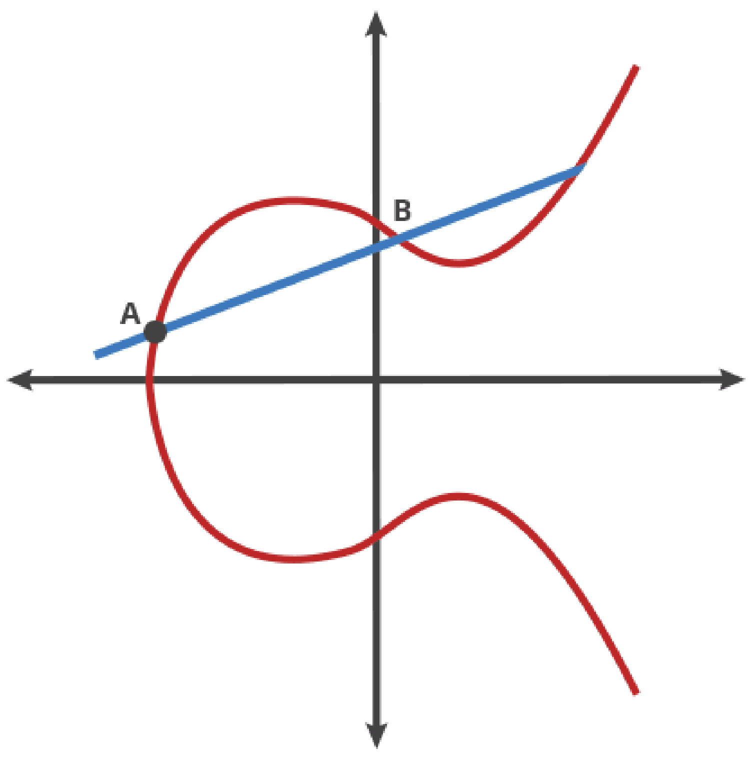
> $$y^2 = x^3 + ax + b$$

The curve would look something like this: (used desmos.com to generate the below diagram)



The above curve has several interesting properties.

One of these is ***horizontal symmetry.*** Any point on the curve can be reflected over the x axis and remain the same curve. A more interesting property is that any non-vertical line will intersect the curve in at most three places.



There is a max threshold which is set, when it goes beyond max value it wraps around to the start of the curve much like modular arithmetic.

The private key in this function would be the number of times we bounce the value before we stop.

It is very difficult to know where it at started.

Comparing it to RSA the computing power needed to break a 228-bit elliptic curve key would be equal to breaking a 2380-bit RSA

> How to work with these curves ?
>
> > Since all the curves can offer are points on the curve, we need to have a mechanism for encoding or hashing our data to these points

```
Some of the well known curves are -- P256, secp256k1, Curve25519, M383, E222, E382
```

Elliptic curves also have to property of additive homomorphism

```
plaintext1, plaintext2

C1, C2 = ellipticCurve.encrypt(plaintext1, publicKey)
C3, C4 = ellipticCurve.encrypt(plaintext2, publicKey)

plaintext = ellipticCurve.decrypt(privateKey, C1 + C3, C2 + C4)

plaintext should be (plaintext1 ++ plaintext2)
```

So exploring this option I came to know although elliptic curves are cool, ***but they don't meet our criteria for commutativity***.

This lead me to look into,

> 💡 A commutative encryption scheme based on ElGamal encryption [Paper]

Key Generation

Alice generates a private key of X1 and a public key of:

$$Y1 = g^{x1}(modp)$$

Bob generates a private key of X2 and a public key of:

$$Y2 = g^{x2}(modp)$$

Encryption

message (M), then Alice will encrypt with:

$$c11 = g^{r1}(modp)$$

$$[r1 = randomNumber]$$

and:

$$c21 = M * Y1^{r1}(modp)$$

Alice will pass c21 to Bob, and who will create:

$$c12 = g^{r2}(modp)$$

$$[r2 = randomNumber]$$

and

$$c22 = c21 * Y2^{r2} (mod p)$$

The encrypted values are c11, c12 and c22

Decrypt (Bob and Alice)

To decrypt, Bob can take his key off with:

$$EncBobRemoved = c22/(c12)^{x2}$$

and then recover the message with:

$$MRecovered = EncBobRemoved/(c11)^{x1}$$

$$MRecovered == M$$