Cryptography

7 – Loose ends

G. Chênevert

November 18, 2019

ISEN ALL IS DIGITAL! LILLE | yncréa

Today

- Elliptic curves
- Key management
- Proofs
- Homomorphic encryption
- And more...

Recall: Generalized DLP

Let $(\mathcal{G}, \cdot\,)$ be a finite abelian group.

Given $g \in \mathcal{G}$ and x such that

$$x = g^{\xi} = \underbrace{g \cdot g \,\cdots\, g}_{\xi} \quad \text{in} \;\; \mathcal{G},$$

find $\xi \underset{\nu}{\equiv} \log_g(x)$, with $\nu = \mathrm{ord}_{\mathcal{G}}(g)$, the smallest $\nu > 0$ for which $g^{\nu} = 1$.

Best known DL algorithm: $\mathcal{O}(\nu^{\frac{1}{2}})$ for a generic group $\mathcal{G}$. (Much smaller for $\mathcal{G} = (\mathbb{Z}/n\mathbb{Z})^{\times}$.)

Elliptic curves

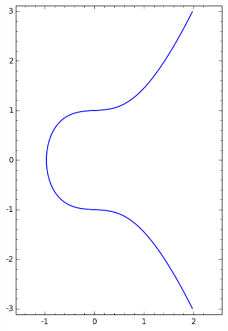
Definition

An *elliptic curve* is a plane curve defined by an equation of the form

$$\mathcal{E}: \ y^2 = x^3 + ax + b.$$

Example

$a = \frac{1}{10}, \ b = 1$



Addition on an elliptic curve

Given $P, Q \in \mathcal{E}$, the line through P and Q intersects $\mathcal{E}$ at a third point, say $R = (x, y)$.

Definition

$$P + Q := (x, -y)$$

Fun fact: This makes $\mathcal{E} \cup \{O\}$ into an abelian group!

(The *point at infinity* $O = (0, \infty)$ being the neutral element)

DLP on an elliptic curve

Given $G \in \mathcal{E}$ of (additive) order n and $P \in \mathcal{E}$ such that

$$P = mG = \underbrace{G + \cdots + G}_{m} \quad \text{in} \;\; \mathcal{E},$$

find $m \underset{n}{\equiv} \log_G(P)$.

(Easy to solve over the real or complex numbers)

Elliptic curves over finite fields

Instead: consider solutions modulo a fixed prime p

$$y^2 \underset{p}{\equiv} x^3 + ax + b$$

$\rightsquigarrow \mathcal{E}(\mathbb{F}_p)$ elliptic curve over the finite field $\mathbb{F}_p$

(a finite abelian group!)

Basic computations are easy…

```
p = 32806352226718822643429
a = 5740347588375554626864
b = 20798093206103976495852

E = EllipticCurve(GF(p),[a,b])

P = E([29155336995917130553754, 8373057744944244479010])
Q = E([3415221595160200314960, 11073266156995522792160])

2*P + 3*Q
```

Evaluate   Language: Sage

Share

```
(9956939019642126506349 : 26680698275736540367982 : 1)
```

Help | Powered by SageMath

…but the DLP is hard!

Size of $\mathcal{E}$

Theorem (Hasse bound)

$$\#\mathcal{E}(\mathbb{F}_p) = 1 + p + \mathcal{O}(\sqrt{p})$$

hence $\#\mathcal{E}(\mathbb{F}_p) \approx p$.

We use elliptic curves with points G of large order $n \approx p$.

ECDH

- Alice and Bob agree on "safe" parameters $\mathcal{E}$ and G.
- Alice chooses a, computes $A = aG$ in $\mathcal{E}$.
- Bob chooooses b, computes $B = bG$ in $\mathcal{E}$.
- Shared secret is

$$K := (ab)G = aB = bA.$$

ECElGamal

Keys:

- d private decryption key
- $E = dG$ public encryption key

Alice wants to send a message $M \in \mathcal{E}$ to Bob.

ECElGamal

Encryption:

- Alice chooses random s, computes $S = sG$
- Computes shared secret $K = sE$
- Computes encrypted $C = M + K$
- Sends the pair (S, C)

Decryption:

Upon reception of a pair (S, C), Bob

- Computes shared secret $K = dS$
- Recovers $M = C - K$

Parameter generation

To get ℓ bits of security:

- choose a 2ℓ-bit prime p
- an elliptic curve $\mathcal{E}$ over $\mathbb{F}_p$
- and a point G on $\mathcal{E}$ of (almost) prime order n that generates (most of) $\mathcal{E}(\mathbb{F}_p)$.

Much harder to manufacture than *e.g.* for RSA – but can be reused.

Recommended curves

In the US, NIST proposed in 2005 a list of 5 elliptic curves of size

$$192,\ 224,\ 256,\ 384 \text{ and } 521 \text{ bits}$$

(as well as 5 curves over binary fields $\mathbf{F}_{2^k}$)

$$\cdots$$

Dual_EC_DRBG controversy

Alternative: Brainpool curves

Also: recent concern about Suite B *cf.* rise of quantum computing!?

Post-quantum cryptography

Ongoing NIST standardization process for quantum-resistant primitives.

Round 2: 17 public-key encryption primitives, 9 digital signature primitives.

Broadly fall into 4 categories:

- lattice-based
- code-based
- hash-based
- multivariate polynomial-based

Stay tuned!

Today

Elliptic curves

Key management

Proofs

Homomorphic encryption

And more...

Key management

Consider a pool of n users, each of which could want to communicate confidentially with any other.

$\implies \binom{n}{2} = \frac{n(n-1)}{2}$ interactions to secure.

With a single secret key for every potential interaction:

every user needs to securely obtain and store $n-1$ *secret* keys!

Purely asymmetric solution

Use public-key encryption for everything.

Every user needs access to any of the $n - 1$ other *public* keys

But: asymmetric ciphers are much slower than symmetric ones.

$\implies$ *hybrid* systems are usually favored (but: full-fledged PKI needed)

Example: TLS/SSL

TLS 1.3 specification

- X.509 certificates are used to authenticate the parties
- A **master secret** is set up
- Bulk of communication encrypted with a symmetric cipher
- MACs are included for data integrity

Various combinations of cipers and MACs (**cipher suites**) are supported (providing varying levels of security).

Cipher suite: example

- RSA-PSS signature for server authentication
- ECDH for key agreement
- Sessions keys are derived from the master secret
- AES-CBC used for encryption
- SHA256-HMAC for message authentication

Agreed upon during initial *handshake*.

Comments

- Provides **forward secrecy** if fresh DH parameters are used every time (recommended!)
- These parameters are signed, preventing man-in-the-middle attacks
- Session keys need to be refreshed after a while
- Often subject to *downgrade attacks*

Kerberos

Purely symmetric key management solution using a trusted **key server** S

Alice wants to communicate securely with Bob.

- Both set up secret keys k_A and k_B with the server.
- Alice asks the server for a secret key k_{AB} to be used with Bob.

Needham-Schroeder algorithm (1978)

- The server replies to Alice with

$$E\big(k_A,\, k_{AB} \,\|\, E(k_B, k_{AB})\big).$$

- Alice decrypts this message and sends to Bob

$$E(k_B, k_{AB}).$$

Alice and Bob now have k_{AB} and can start communicating securely.

Comments

- Nonces need to be included to prevent *replay attacks*
- Provides mutual authentication as well as confidentiality
- Man-in-the-middle attacks are not possible
- Server does not need to remember keys
- But: single point of failure

Today

Elliptic curves

Key management

Proofs

Homomorphic encryption

And more...

How to trust others' computations?

In various cryptographic protocols, Bob might worry that Alice is not doing things properly

(read: cheats! – or makes mistakes)

and ask her for *proofs* of good conduct.

Bob: *challenger*

Alice: *prover*

Infamous example: proof of work

To make sure that Alice has access to suitable computing resources:

on input m, asks her to find a string k for which the binary representation of

$$H(m \,\|\, k) \text{ starts with } n \text{ zeros.}$$

Partial collision problem: her best approach is to brute-force k

will take 2^n trials on average

(this is what Bitcoin cryptominers do... with an ecological impact of epic proportions)

Example: coin flip

Alice and Bob play a game.

Heads: A gives €100 to B, tails: B gives €100 to A.

Alice is responsible for tossing the coin.

Alice: "Tails!"

Bob: "Prove it!"

Secure coin flip

- Alice chooses a random large integer n
- Sends its SHA256 hash to Bob (*commitment*)
- Bob selects $b \in \{0, 1\}$, sends it to Alice
- Alice returns $(n \% 2) \oplus b$ (result of coin toss)

  and n (proof of randomness)

Alice cannot manipulate the result unless she knows n and n' of different parity with the same hash!

Zero-knowledge proofs

Sometimes Alice wants to convince Bob of a certain statement, *without revealing anything else than the fact that this statement is true.*

Example

Alice: "I know ξ such that $g^{\xi} \underset{p}{\equiv} x$"

Bob: "Prove it!"

Zero-knowledge proof

Idea: Bob should present Alice with requests that she can only answer correctly if she does indeed know ξ – and that Bob can check are answered correctly.

- Alice chooses a random number $\rho \in]\!]0, q[\![$ and sends $c \underset{p}{\equiv} g^{\rho}$ to Bob.
- Bob randomly requests Alice to either disclose

$$\rho \qquad \text{or} \qquad \rho + \xi \qquad \mod q.$$

Correctness

If Bob receives exponent ρ' from Alice, he can check the agreement with *commitment* c by computing

$$g^{\rho'} \qquad \text{or} \qquad g^{\rho'} \cdot x^{-1} \qquad \bmod p.$$

Alice can easily fake a correct answer (without knowing ξ) to any of those questions *but not both*. She would have to guess correctly which question Bob will ask before to commit an adequate value of c.

If Alice answers correctly n requests in a row, Bob can trust that the probability that she knows ξ is $\geq 1 - \frac{1}{2^n}$.

Today

Elliptic curves

Key management

Proofs

Homomorphic encryption

And more...

Malleability, revisited

We mainly considered malleability a bad thing.

But it can actually be useful!

Example

Alice wants to compute the product of two ℓ-bit integers m_1 and m_2. She could

- Encrypt them using plain-RSA with a 2ℓ-bit modulus
- Send the ciphertexts to Bob and ask *him* to multiply them
- Decrypt the resulting ciphertext.

Homomorphic encryption

Certain ciphers preserve addition *or* multiplication.

Definition

A **fully homomorphic** cipher is one that preserves both addition and multiplication.

So what?

A cryptographer's dream

1978

Suppose we have a fully homomorphic cipher

$$E : \mathcal{M} = (\mathbf{F}_2, \oplus, \odot) \longrightarrow \mathcal{C}.$$

Then, since

$$\begin{cases} x \texttt{ and } y = x \odot y \\ x \texttt{ or } y = x \oplus y \oplus (x \odot y) \\ \texttt{not } x = 1 \oplus x \end{cases}$$

we can build a processor that works with encrypted bits!

Fast-forward to 2009

Theorem (C. Gentry, Standford Ph.D. thesis)

Fully homomorphic ciphers exist.

Gentry's original construction used lattice-based cryptography but a more elementary one was later found.

In both approaches, one starts with a *somewhat homomorphic cipher*.

Somewhat homomorphic encryption

Secret key: a large odd integer k

Encryption: to encrypt $b \in \{0, 1\}$, choose random q and m with $2m \in [\![0, k-1[\![$ and set

$$c = qk + 2m + b.$$

Decryption: $b = (c \,\%\, k) \,\%\, 2$

Bootstrapping

These encrypted bits can support a limited number of operations while still decrypting correctly.

After that: need to refresh encryption.

How to do that in the blind processor?

Decrypt through the encryption!

Refreshing encryption

- Alice sends $c_1 = E(k_1, b)$ to Bob
- Bob computes $c_{12} = E(k_2, c_1)$
- Then computes $c_2 = D(k_1, c_{12})$ *through the encryption* in order to get

$$c_2 = E(k_2, b).$$

(For this to work, an asymmetric version of the cipher needs to be used)

A somewhat homomorphic cipher only needs to support its own decryption circuit *plus one operation*.

So is this used everywhere in the cloud?

Not yet. . . still an area of active research & development.

Current implementations are still somewhat impractical (slow / large keys)

One could in principle run arbitrary encrypted code on arbitrary encrypted data on a remote processor and get the encrypted result back!

Today

- Elliptic curves
- Key management
- Proofs
- Homomorphic encryption
- And more...

And more...

- split secrets
- secure multipartite computation
- identity and attribute-based encryption
- digital currencies (blockchain)
- differential privacy
- quantum cryptography

New Crypto Wars episode coming soon to a computer near you . . .