Corso Professionalizzante di Specializzazione (3 CFU)

Ingegneria dell’Informazione o magistrale in Ingegneria Informatica Automatica, Ingegneria Elettronica, Ingegneria delle Telecomunicazioni

WSN and VANET Security

Part II: Techniques for WSN and VANET Security

Lecture II.1

Passive Security Functions: Mathematical Background

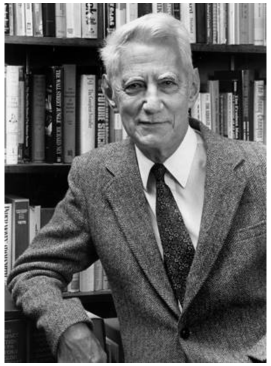
Ing. Marco Pugliese, Ph.D., SMIEEE
Senior Security Manager cert. UNI 10459-2017
marpug@univaq.it

April 26th, 2024

Kerckhoffs’ Principle

- The dutch cryptographer A. Kerckhoffs (1835-1903) stated the design principles for military ciphers (*La Cryptographie Militaire, 1883)*
- **Kerckhoffs' principle**: «A cryptosystem should be secure even if everything about the system, except the key, is public knowledge, and it should not be a problem if it falls into enemy hands».
- This in contrast to **security through obscurity.**
- Kerckhoffs viewed cryptography as a better alternative than **steganographic encoding**, which was common in the nineteenth century for hiding the meaning of military messages.
- The american mathematician and engineer C. E. Shannon (1916 – 2001) has been the father of Information Theory and the first to guess that security was a matter from information theory (information theoretic security).

The Shannon’s Lessons

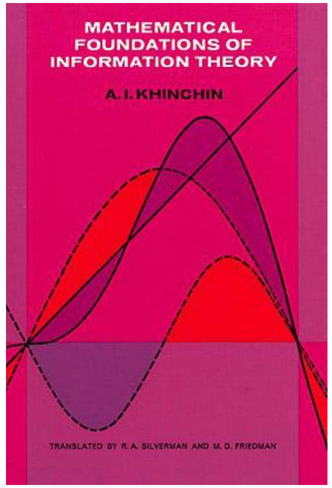


C. E. Shannon, **A Mathematical Theory of Communication**
Bell System Technical Journal, vol. **27** (3): 379–423, July 1948
http://www.essrl.wustl.edu/~jao/itrg/shannon.pdf

C. E. Shannon, **Communication Theory of Secrecy Systems**
Bell System Technical Journal, vol. **28** (4): 656–715, October 1949
http://netlab.cs.ucla.edu/wiki/files/shannon1949.pdf

Claude Elwood Shannon, 1916-2001

The sovietic mathematician **Aleksandr Jakovlevič Chinčin** (1894-1957) with his **Mathematical foundations of information theory,** gave the first comprehensive introduction to information theory, places the work begun by Shannon and continued by McMillan, Feinstein on a rigorous mathematical basis. Chinčin develops the concept of **entropy** in probability theory **as a measure of uncertainty** of a finite scheme, and discusses a simple application to coding theory, and investigates the restrictions previously placed on the study of sources, channels, and codes and attempts “to give a complete, detailed proof of both Shannon theorems, assuming any ergodic source and any stationary channel with a finite memory.”



A compendium of C.E. Shannon and A.J. Chinčin’s masterworks with application to cryptography can be found in:
M. Pugliese, Fundamentals of Information Theory with Application to Cryptography - Summary from the lessons by C.E. Shannon and A.Y. Khinchin, v.3.4, 2021, https://mpugliese.webnode.it/scientific-contributions/

Lesson 1: Shannon Entropy

- Classical **entropy** is a measure of disorder in a system: disorder refers to the uncertainty about the determination of the hidden particular micro-configurations that correspond to the observable macro-configuration. **Shannon entropy H(P)** is a measure of uncertainty about a discrete stochastic process P:

  $\mathbf{H(P) = -\sum_P pr(p)\log_2(pr(p))}$ where **p is a determination of the process P** and pr(p) the probability of that determination where $0 < pr(p) \leq 1$.
- **Shannon-Chinčin : information corresponds to uncertainty,** i.e. **I(P) = H(P).**
- Determinations from a **known** process P (then pr(p)=1) imply **H(P) = 0**, i.e. **no uncertainty about P**, and imply **I(P) = H(P) = 0**, i.e. **no added information about P**.
- **Conditional Shannon entropy H(P|Q)** is a measure of the uncertainty about the process P once known a determination of another process Q.

  $\mathbf{H(P|Q) = -\sum_{P,Q} pr(p|q)\log_2(pr(p|q))}$ where **pr(p|q)= pr(p,q)/pr(q)** (Bayes)

  Therefore **H(P|Q) = H(P,Q)-H(Q)**;

  Iff P and Q are **statistically independent** then **H(P|Q) = H(P)**, hence H(P,Q) = H(P)+H(Q). Otherwise **H(P|Q) < H(P)**, hence H(P,Q) < H(P)+H(Q).
- **I(P;Q) = H(P) - H(P|Q) ≥ 0**: defines the information about P with Q given by the uncertainty about P **without** the knowledge of Q reduced by the uncertainty about P **with** the knowledge of Q. Iff P and Q are **statistically independent** then **I(P;Q) = 0**.

Lesson 1: Shannon Entropy

- Shannon entropy is measured in the unit "**bit**".
- Shannon introduced the operator $\log_2()$ (log base 2) in the definition of entropy for a direct application to digital communications: two logic states or **binits** 0 / 1 associated to two electronic states.
  - In a (honest) coin tossing process, the determinations of coin faces are **equiprobable and statistically independent (p_H=0,5 p_T=0,5)**
  - Which is the entropy associated to a coin tossing (ct) process?

    $H(ct) = -(p_H \log_2(p_H)+p_T \log_2(p_T)) = -2(1/2)\log_2(1/2) =$ **1 bit**.
- If the "**emission of a random sequence of binary digits**" process is statistically equivalent to a "coin tossing" process: the generated bitstream is a **random sequence** and the **entropy of the process = 1 bit / binit**.
- Truly random bitstreams cannot be inherently generated by whatever **deterministic** algorithms or process even in the case of high entropy seed (only a stochastic algorithm or process can generate truly random bitstreams. Therefore only **pseudorandom bistreams** can be available (entropy < 1 bit / binit).
- Given a generic stochastic process P, the **upper bound** for H(P) is $\mathbf{\log_2|P|}$ which corresponds to the entropy if P were a random process:

  $\mathbf{H(P) \leq \log_2|P|}$ where $\log_2|P| = -\sum_{|P|}(1/|P|)\log_2(1/|P|)$ by setting $pr(p)=1/|P|$.

Lesson 2: Secrecy Classification

- A function is **computationally infeasible** if its time complexity **is more than polynomial time (e.g. sub-exponential or exponential time)**: an algorithm is polynomial time (or has polynomial time complexity) if for some m, C > 0, its running time (dependent on the available computational resource) on inputs of size n is at most Cn^m or, equivalently, an algorithm is polynomial if for some m > 0 its running time on inputs of size n is $O(n^m)$.
- The value of the "input size" depends on the nature of the problem: in the case of cryptosystems, size n is the order of the reference finite field.
- A function f(x) is **one-way** (or **surjective** or **many-to-one** or **with collisions**) if complexity of y = f (x) is **polynomial time** and $x = f^{-1}(y)$ is **computationally infeasible**.
- Given k, a function $f_k(x)$ is **one-way** if complexity of $y = f_k(x)$ is **polynomial time** and $x = f_k^{-1}(y)$ is **computationally infeasible if k unknown** or **polynomial time if k known**. $x = f_k^{-1}(y)$ is also denoted as the **reverse cryptographic problem**.
- The reverse function $x = f^{-1}(y)$ or **$f_k^{-1}(y)$ with k unknown** are palindrome (or **one-to-many**) and **spurious solutions** can result.
- The reverse function **$f_k^{-1}(y)$ with k known** is **invertible** (or **one-to-one**).
- Many cryptographic functions are one-way functions: e.g. cryptographic secure pseudo random generators, RSA encryption / decryption function, block ciphers, discrete logarithm problem, hash function, square root, …

Lesson 2: Secrecy Classification

Let P the process "**emission of a sized sequence of binary digits**" and C the process **"computation of $C=f_k(P)$"** where $f_k()$ is a one-way function and k is the key.

- **Perfect (or unconditional) Secrecy**: the uncertainty on plain-text is not reduced by the observation of the related cipher-text.

  Once known the cipher-text, the uncertainty of the plain-text **is equal to** the uncertainty of the plain-text unknown the cipher-text (P and C are statistically independent)

$$H(P|C) = H(P)$$

  Therefore **no information** is gained from the knowledge of the cipher-text.

$$I(P;C) = H(P) - H(P|C) = 0$$

- **Realistic (or conditional) Secrecy**: if the uncertainty on plain-text is reduced by the observation of the related cipher-text.

  Once known the cipher-text, the uncertainty of the plain-text **is less than** the uncertainty of the plain-text unknown the cipher-text.

$$H(P|C) < H(P)$$

  Therefore **some bit of information** is gained from the knowledge of the cipher-text.

$$I(P;C) = H(P) - H(P|C) > 0$$

Lesson 3: Perfect Secrecy

- Let K,P,C be instances of the same process "**emission of a sized sequence of binary digits".**
- Let **|K|,|P|,|C|** be the **number** of sized sequences of binary digits (bistrings) that K, P, C can emit.
- Let $len(K)=\log_2|K|$, $len(P)= \log_2|P|$, $len(C)=\log_2|C|$ be the lengths of the generic sized sequence of binary digits that K, P, C can emit.
- Let $k \in \{0,1\}^{len(K)}$ $p \in \{0,1\}^{len(P)}$ $c \in \{0,1\}^{len(C)}$ be generic sized sequences emitted by K, P, C. A bistring emitted by P and C are also called **block** or **gram.**
- Let $e_k() \in E$ be an encryption **one-way function** with key k such that for $\forall p \in P$ and any k is $c = e_k(p)$.

Shannon in his "Communication Theory of Secrecy Systems", introduced the following fundamental theorem:

- **Theorem on Perfect Secrecy:** suppose a cryptosystem where $|K| = |C| = |P|$. Then the cryptosystem provides **perfect secrecy** if and only if any key k is used with equal probability $1/|K|$ and $\forall p \in P$, there exists a **unique key k** $\in$ K and $c \in C$ such that $e_k(p) = c$. Therefore for $\forall p \in P$ and any $k \neq k'$ is $e_k(p) \neq e_{k'}(p)$.

Lesson 4: Key and Message Equivocation

Suppose a "brute force" attacker is observing a transmitted ciphertext.

- **Theorem on Key Equivocation**: the amount of uncertainty (or equivocation) on the key that remains after knowing the cipher-text, indicated with H(K|C), is given by:

$$H(K|C) = H(P) + H(K) - H(C)$$

- **Key Equivocation is a performance index for a cryptosystem**: it should be as larger as possible. Be C_n the n-th cipher-text block (n-gram) observed by the attacker:
- Upper bound is $\mathbf{H(K|C_n)=H(K)}$ as $H(C_n)_{min} = H(P_n)$.
- Lower bound is $\mathbf{H(K|C_n)=H(K)-nR_P\log_2|P|}$ where R_P is the redundancy of the plain-text:

$$R_P = 1 - \frac{H(P)}{\log_2|P|}$$

For large n, if $R_P \to 0$ then $H(K|C_n) \to H(K)$, hence Key Equivocation gets its upper bound.

Lesson 5: Spurious Keys

- To a given Key Equivocation H(K|C) corresponds a set of keys (denoted as **Spurious Keys**) for which the cipher-text can deciphered in multiple plain-texts (remember that an encryption function with unknown key is one-to-many) excepting the legitimate ciphering key.
  Let s_n be the **expected number of spurious keys** corresponding to the n-th cipher-text block observed by an attacker, Shannon showed that:

$$s_n = \frac{|K|}{|P|^{nR_P}} - 1$$

  With increasing n, Spurious Keys reduce.
- It is important to determine the **minimum n_0** for which the (expected) number of spurious keys should be **zero** (only the legitimate key is expected to remain).

Observation: an attacker should record at least n_o binits of cipher-text to expect to solve **univocally** the cryptographic reverse problem on that cipher-text shall produce the only legitimate key. Therefore: **n_o should be as larger as possible.**

Lesson 6: Unicity Distance

- The number n_0 is called **Unicity Distance**.
- Let impose $s_n = 0$ for $n=n_o$

$$n_0 = \frac{\log_2|K|}{R_P \log_2|P|}$$

If $R_P \rightarrow 0$ and/or if $|K| >> |P|$, then n_0 gets larger.

- Therefore reduced redundancy (high compressions) and large space key enhance communication robustness **from a cybersecurity viewpoint**.
- The object of coding is designing efficient and reliable data transmission methods. This typically involves the removal of redundancy (**source coding**) and the correction / detection of errors in the transmitted data (**channel coding**) to enhance communication robustness **from a noisiness viewpoint** …
- … but correction / detection of errors introduces some code redundancy!

Need to find a balance.

Outline

- Modular Arithmetic
- Generating Prime Numbers
- Generating Pseudo-random Numbers
- Elliptic Curve Algebra
- Discrete Logarithm Problem and its EC version
- Pairings on Elliptic Curves
- Zero Knowledge Proof

Finite Groups

- A **finite group** G(n,$\circ$) is a set of n elements and one operation symbolically denoted with $\circ$.
- Operation $\circ$ satisfies four group axioms: **closure, associativity, identity (0) and invertibility.**
  - **closure:** $\forall a,b \in G \,/\, a \circ b \in G$
  - **associativity:** $\forall a,b,c \in G \,/\, a \circ (b \circ c) = (a \circ b) \circ c$
  - **identity:** $\forall a \in G: \exists!\, \mathbf{0}$ (zero) $/\, a \circ \mathbf{0} = a$

    0 = identity respect to $\circ$
  - **invertibility:** $\forall a \in G: a \circ (-a) = \mathbf{0}$

- If also **commutativity** then the group is **abelian.**
  - **closure:** $\forall a,b \in G \,/\, a \circ b = b \circ a \in G$
  - **associativity:** $\forall a,b,c \in G \,/\, a \circ (b \circ c) = (a \circ b) \circ c = (b \circ c) \circ a = b \circ (c \circ a)$
  - **identity:** $\forall a \in G: \exists!\, \mathbf{0}$ (zero) $/\, a \circ \mathbf{0} = \mathbf{0} \circ a = a$

    0 = identity respect to $\circ$
  - **invertibility:** $\forall a \in G: a \circ (-a) = (-a) \circ a = \mathbf{0}$

Finite Groups

- **Order of a finite group** or ord(G) = the number of elements of a finite group
- **Order of an element a of a finite group**: ord(a)=n if n is the smallest integer such that a ◦ a ◦ … (n times) … ◦ a = **0**.
- A group G(n,◦) is a **cyclic group** if all n elements **can be generated from a single element by applying iteratively the defined operation** ◦.
  - This element is called **base element (**or **generator)** of the group respect to the operation ◦.
  - The **order of a cyclic group**, is also called the **order of the generator.**

- A **subgroup** of a group is a subset of the elements of the group for which still holds the definition of group. The number of elements of a subgroup determines **the order of the subgroup**.
- A **cyclic subgroup** of a cyclic group is a subset of the elements of the cyclic group for which still holds the definition of cyclic group. The number of elements of a cyclic subgroup determines **the order of the cyclic subgroup**.

The example G(10,+)

- Suppose G(10, +) additive abelian group of integers 0, 1, 2, …, 9. Let a, b ∈ G.
- Operator **+** is defined as follows: **a + b = remainder of (a+b)/n (a + b mod n)**
- Is G a cyclic group? Yes, because:

  1, 1+1=2, 2+1=3, 3+1=4, 4+1=5, 5+1=6, 6+1=7, 7+1=8, 8+1=9, 9+1=0: **1** is a generator

  3, 3+3=6, 6+3=9, 9+3=2, 2+3=5, 5+3=8, 8+3=1, 1+3=4, 4+3=7, 7+3=0: **3** is a generator

  7, 7+7=4, 4+7=1, 1+7=8, 8+7=5, 5+7=2, 2+7=9, 9+7=6, 6+7=3, 3+7=0: **7** is a generator

  9, 9+9=8, 8+9=7, 7+9=6, 6+9=5, 5+9=4, 4+9=3, 3+9=2, 2+9=1, 1+9=0: **9** is a generator
- In general for an additive cyclic group order n, the element k is a generator iff **gcd (k,n)=1**, or k and n are co-primes; if **gcd (k,n)>1** then k is a generator of a subgroup of order n/gcd; the number of subgroups is equal to the number of divisors of the group order n.
- Therefore: 1,3,7,9 are generators of G(10,+); 2 subgroups say A and B order 5 and 2; 2,4,6,8 are generators of subgroup A and 5 is generator of subgroup B.

  gcd(2,10)=2; order = 10/2=5

  gcd(4,10)=2; order = 10/2=5

  gcd(5,10)=5; order = 10/5=2

  gcd(6,10)=2; order = 10/2=5

  gcd(8,10)=2; order = 10/2=5
- If **n prime**, any element in G is generator of G because **gcd (∀k,n)=1**, **no subgroups** because n has no divisors (n is prime).

1	2	3	4	5	6	7	8	9	0
2	4	6	8	0	2	4	6	8	0
3	6	9	2	5	8	1	4	7	0
4	8	2	6	0	4	8	2	6	0
5	0	5	0	5	0	5	0	5	0
6	2	8	4	0	6	2	8	4	0
7	4	1	8	5	2	9	6	3	0
8	6	4	2	0	8	6	4	2	0
9	8	7	6	5	4	3	2	1	0

Finite Fields (Galois fields)

- An abelian **finite field** (or Galois field) GF(G, *) extends a finite abelian additive group G(n, +) by adding a further operation * and the further group axiom **distributivity**:
  - **closure:** $\forall a,b \in GF$ / $a + b \in GF$, $a * b \in GF$
  - **associativity:** $\forall a,b,c \in GF$ / $a + (b + c)=(a + b) + c$, $a * (b * c)=(a * b) * c$
  - **identity:** $\forall a \in GF$: $\exists!$ **0** (zero) / $a + \mathbf{0} = a$, $\exists!$ **1** (one) / $a * \mathbf{1} = a$

    0 = additive identity, **1** = multiplicative identity
  - **invertibility:** $\forall a \in GF$: $a + (-a) = \mathbf{0}$, $a * (a^{-1}) = \mathbf{1}$
  - **distributivity** of * respect to +: $\forall a,b,c \in GF$ / $(a + b) * c=(a * c) + (b * c)$

- **Characteristic of GF** or char(GF): char(GF)=k if k is the smallest integer such that **1** + **1** + … (k times) … + **1** = **0**, otherwise char(GF)=0.
- **Order of a finite field** or ord(GF) = the number of elements of a finite field.
- A finite field of order q exists **if and only if the order is $q = p^k$** where **p is a prime** and **k is a positive integer**. Therefore we can only have GF(p), GF(p^2), …, GF(p^k).
- GF(p) is denoted as “ordinary” GF, GF(p^k) are denoted as “Galois field extensions”
- Char(GF($q= p^k$)) = p
- Let’s start with GF(p)

Operations with ordinary Galois fields

Given $a,b \in GF(p)$, define the operations + and * as follows:

- The **addition (+)** is defined as the remainder of $(a+b)/p$ ($a + b \bmod p$)
- The **product (*)** is defined as the remainder of $(a*b)/p$ ($a * b \bmod p$)
- The **additive inverse** (**opposite**) of a (indicated with -a) is defined as p-a
- The **multiplicative inverse** of a (indicated with a^{-1}) is
  - computed using the Extended Euclidean Algorithm.
  - computed using the Fermat Little Theorem.

- It can be shown that non-zero elements of an **ordinary** Galois field form a multiplicative cyclic group.
- Let us search for generators of GF(p).
- It can be shown that given $g \in GF(p)$, then **g is a generator of GF(p)** if $g^{(p-1)/q} \neq 1$ where q is a prime divisor of p-1.

  E.g. p=7: prime divisors of 6 (=7-1) are q=2 and q=3.

  g=3 and g=5 are generators because $3^3 \neq 1$, $3^2 \neq 1$ and $5^3 \neq 1$, $5^2 \neq 1$.

  g=3: $3^1=$**3**, $3^2=$**2**, $3^3=$**6**, $3^4=$**4**, $3^5=$**5**, $3^6=$**1**

  g=5: $5^1=$**5**, $5^2=$**4**, $5^3=$**6**, $5^4=$**2**, $5^5=$**3**, $5^6=$**1**

Extended Euclidean Algorithm

- The Euclidean Algorithm computes the "*greatest common divisor*" (gcd) between a couple of integers a and b.
- The *Extended* Euclidean Algorithm computes the "*greatest common divisor*" (gcd) between a couple of integers a and b, and computes the coefficients x and y of the so called "Bézout's identity":

$$\mathbf{ax+by = gcd(a,b)}$$

- If b=p (p prime), then a and p are co-primes, thus gcd(a,p)=1:

$$\mathbf{ax+py = 1}$$

- It can be easily shown by applying **modulo** p in both terms, and being **py mod p = 0** $\forall$y, we get **ax = 1 mod p**.

- Therefore x is the modular multiplicative inverse modulo p of a

$$\mathbf{x = a^{-1} \ mod\ p.}$$

Fermat Little Theorem

Theorem: given p prime in GF(p), $\forall a \neq 0$ is

$$\mathbf{a^{(p-1)} \bmod p = 1}$$

Corollaries:

- Multiplying by a both terms is **a^p mod p = a** (**cyclicity**).

- Multiplying by a^{-1} both terms is **$a^{(p-2)}$ mod p = a^{-1}** (**inverse**).

- Example in GF(7): $\forall a \neq 0$
  - $a^6 \bmod 7 = 1$
  - $a^7 \bmod 7 = a$
  - $a^{-1} \bmod 7 = a^5 \bmod 7$

- Therefore inversion through exponentiations.
- Exponentiation is energy and time consuming: these are some algebraic tricks (e.g. Square and Multiply algorithm) to minimize computations.
- Generally the Extended Euclidean Algorithm to be preferred in terms of complexity.

Extended Galois Field GF(p^n)

- GF(p^n) extends GF(p) and is an abelian cyclic group with p prime, n integer.
- Note that **p^n is never prime**.
- **Elements in GF(p^n) are p^n polynomials degree up to n-1 with n coefficients in GF(p)**: therefore elements in GF(p^n) are p^n n-plas in GF(p) and Char(GF(p^k)) = p.
- **Special case p = 2** $\rightarrow$ GF(2^n): coefficients in GF(2), i.e. booleans.

  E.g. GF(2^3): 8 elements: **0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1**

  8 3-plas: **000, 001, 010, 011, 100, 101, 110, 111**

  For GF(2^n): 2^n n-plas (all combinations of 2 elements in groups of n).

  Easy costruction of G(p^n): any bit string size n is an element in GF(p^n)
- **Irreducible polynomial**: a polynomial p(x) **degree n** divisible only by 1 and by itself. It is used for congruences (the same as mod p in GF(p)).

  E.g. GF(2^3): an irreducible polynomial is p(x)=x^3+x+1, notation is GF(2^3)/x^3+x+1.

  E.g. GF(2^8): **GF(2^8)/x^8+x^4+x^3+x+1** (Rijndael polynomial in AES).
- Computing irreducible polynomials is an advanced topic (Artin–Schreier theory).

Operations in GF(2^n) as well as operations with bit strings size n correspond to congruence operations between polynomials degree up to n-1.

EXEMERGE CELLENCE

Operations with extended Galois fields

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Operations on polynomials in $GF(p^n)$ corresponds to operations on their coefficients in GF(p). Given $a,b \in GF(p^n)$, define the operations + and * as follows:

- The **addition** (**+**) is defined as a+b
- The **product** (*****) is defined as the remainder of (a*b)/p(x)   (a * b mod p(x))
- The **additive inverse** (**opposite**) of a (indicated with -a) is defined as the polynomial where each coefficient is the additive inverse in GF(p)
- The **multiplicative inverse** of a (indicated with a^{-1}) is
  - computed using the Polynomial Extended Euclidean Algorithm.
  - computed using the Fermat Little Theorem.

- It can be shown that non-zero elements of an **extended** Galois field form a multiplicative cyclic group.
- Let us search for generators of $GF(p^n)$.
- It can be shown that given $g \in GF(p^n)$, then **g(x) is a generator of $GF(p^n)$** if $g^{(p^n-1)/q} \neq 1$ where q is a prime divisor of p^n-1.

  E.g. **p=2**, n=3, $p^n=2^3$: prime divisors of 7 (=8-1) is only q=7. Hence any $g(x) \in GF(2^3) \neq 1$ is a generator.

  E.g. g(x)=x: $x^1=\mathbf{x}$, $x^2=\mathbf{x^2}$, $x^3=\mathbf{x+1}$, $x^4=\mathbf{x^2+x}$, $x^5=\mathbf{x^2+x+1}$, $x^6=\mathbf{x^2+1}$, $x^7=\mathbf{1}$

Operations with GF(2^n)

Operations on polynomials in GF(2^n) corresponds to operations on their coefficients in GF(2):

- <u>**Addition**</u>**:** 0+0 mod 2 = 0; 0+1 mod 2 = 1; 1+0 mod 2 = 1; 1+1 mod 2 = 0
  Hence 0+0 = 0; 0+1 = 1; 1+0 = 1; 1+1 = 0
  This is equivalent to a XOR between coefficients.
- **Substraction:** a-b = a+(-b) mod 2
  <u>**Opposite**</u>: -a=2-a mod 2: -0 = 2-0 mod 2 = 0; -1 = 2-1 mod 2 = 1
  Hence 0+(-0) = 0; 1+(-1) = 0; in general a+(-a) = 0
  Still equivalent to a XOR between coefficients.
  <u>Therefore substraction and addition are coincident operations</u>.
  <u>Hence **any polynomial in GF(2^n) coincides with its opposite**</u>**.**

- <u>**Product**</u>**:** ordinary product between polynomials and, **if the resulting polynomial degree is ≥ irreducible polynomial degree**, then **reduction** by the irreducible polynomial (the same as modulo operations).

- <u>**Division**</u>**:** a/b = a*(1/b) (where 1/b is the <u>**Multiplicative inverse**</u> of b)

Operations with GF(2^n)

- Addition in GF(2^3):
  (110) XOR (101) = (011)
  $(x^2+x)+(x^2+1)=\mathbf{x+1}$

- Product in GF(2^3):
  (110) · (101) mod (x^3+x+1)
  $(x^2+x)\cdot(x^2+1) \bmod (x^3+x+1) = (x^4+x^3+x^2+x) \bmod (x^3+x+1)$

  The degree (4) of product polynomial is greater than the degree (3) of the irreducible polynomial $\rightarrow$ **reduction operation**

  A reduction is an ordinary polynomial division where the irreducible polynomial is the divisor.

  Reduction by the irreducible polynomial x^3+x+1:

  $x^4+x^3+x^2+x = (x+1)(x^3+x+1) + (1)$

  Therefore the product is **1** = (001)

Polynomial Extended Euclidean Algorithm

- The greatest common divisor of two polynomials is a polynomial of the highest possible degree that is a factor of both the two original polynomials (the concept is analogous to the greatest common divisor of two integers).
- Similarly, the Polynomial Extended Euclidean Algorithm computes the multiplicative inverse in algebraic field extensions and, in particular, in finite fields of non prime order (p^n is never prime).

- Polynomial Extended Euclidean Algorithm computes the polynomial greatest common divisor and the coefficients of Bézout's identity of two univariate polynomials.

If a and b are two nonzero polynomials, then the Polynomial Extended Euclidean Algorithm produces the unique pair of polynomials (s, t) such that as+bt=gcd(a,b)

Outline

- Modular Arithmetic
- Generating Prime Numbers
- Generating Pseudo-random Numbers
- Elliptic Curve Algebra
- Discrete Logarithm Problem and its EC version
- Pairings on Elliptic Curves
- Zero Knowledge Proof

Strong Primes

- The number of primes < x is given (with good approximation) by **x/lnx**.
- x/lnx is monotonically increasing for x $\rightarrow \infty$ (primes are infinite).

- Requirements for Strong Primes:
  - gcd(p-1,q-1) is small (important if the key is the product of p with q)
  - Both p-1 and q-1 have large prime factors p', q'
  - Also p'-1 and q'-1 have large prime factors
  - (p-1)/2 and (q-1)/2 are both prime

- **Mersenne Primes** are primes of the form $M_n = 2^n - 1$ for some integer n.
- **Pseudo-Mersenne Primes** are *primes* of the form $2^n - k$, where k is an integer for which $0<|k|<2^{(n/2)}$. **Pseudo-Mersenne and Mersenne primes are useful in cryptography because they admit fast modular reduction**.
- **Safe primes** are *primes* of the form $2p + 1$, where p is also a prime (p is denoted Sophie Germain prime). These primes are "safe" because of their relationship to strong primes: for a safe prime q = 2p + 1, the number q − 1 = 2p has the large prime factor p and so a safe prime q meets part of the criteria for a Strong Prime.

Primality Testing

- AKS Algorithm (Agrawal Kayal Saxena, 2002) is a **deterministic** primality proving algorithm which determines whether a number is prime or composite within **polynomial time.**
- It is applicable to any integer.
- It is not pre-conditioned by any conjecture.

$$O((\log_2 n)^{12})$$

The AKS primality test is based upon the following theorem: **An integer *n* (≥ 2) is prime if and only if the polynomial congruence relation**

$$(x + a)^n = (x^n + a) \quad \mathrm{mod}\, n$$

holds for a, n such that GCD (a,n)=1 (*a* coprime to *n*); *x* is a free variable.

The authors received the 2006 Gödel Prize and the 2006 Fulkerson Prize for this work.

Outline

- Modular Arithmetic
- Generating Prime Numbers
- Generating Pseudo-random Numbers
- Elliptic Curve Algebra
- Discrete Logarithm Problem and its EC version
- Pairings on Elliptic Curves
- Zero Knowledge Proof

Random vs. Pseudo Random Functions

- A **random** number is a number generated by a Random Function (RF) that **cannot** be predicted with any better probability than a random probability distribution before it is generated: e.g. if the number is generated within the range [0, N-1], then its value cannot be predicted with any better probability than 1/N.
  - A **Random Function** (**RF**) is a function f: $\{0, 1\}^n \rightarrow \{0, 1\}^n$ constructed as follows: for each $x \in \{0, 1\}^n$ pick a random $y \in \{0, 1\}^n$ and let f(x)=y.

- A **pseudo-random** number is a number generated by a Pseudo Random Function (PRF) that in line of principle **can** be predicted before it is generated.
  - A **Pseudo-Random Function** (**PRF**) is a function f_s such that $\mathbf{Pr(\{s \leftarrow \{0, 1\}^n: f_s\}_n) - Pr(\{f \leftarrow RF_n: f\}_n) \leq \varepsilon(n)}$ is arbitrarily small.

    Hence f_s defined as the uniform sampling of s from the set $\{0, 1\}^n$ and f defined as the result of a uniform sampling from the set of RF_n are *equivalent,* i.e. probability distributions differ for an arbitrarily small $\varepsilon(n)$.
  - PRF is realized as **a deterministic algorithm** initiated by a **single sample** (**seed**) picked from a **high entropy process**: refer to NIST Special Publication 800-90A / 90B for the requirements of entropy and the related tests.

CSPRNG

A Cryptographically Secure PRNG (CSPRNG) is a PRNG but the reverse is not necessarily true. Requirements are both statistic and cryptologic.

Statistic Test:

- Every CSPRNG should satisfy the **next-bit test**: given the first i bits of a sequence of k bits, there is no polynomial-time algorithm that can predict the (i+1)th bit with probability of success better than 50%.

Cryptologic Test:

- After an attacker has observed "many" previous outputs from the PRNG:
  - It is **computationally infeasible** to compute the internal state of the PRNG.
  - It is **computationally infeasible** to compute the next output of the PRNG.

Cryptographically Secure PRNGs

- RSA Generator
- Blum-Micali Generator
- Blum-Blum-Shub Generator

Outline

- Modular Arithmetic
- Generating Prime Numbers
- Generating Pseudo-random Numbers
- Elliptic Curve Algebra
- Discrete Logarithm Problem and its EC version
- Pairings on Elliptic Curves
- Zero Knowledge Proof

Elliptic Curve Algebra in Cryptography

- Currently public key systems (e.g. RSA) are based on finite field GF(p), p prime, with minimum key length k= 2048 bits, $p \approx 2^{2048}$ and $f_k^{-1}(p)$ its reverse cryptographic problem.

- Neal Koblitz in 1985 observed that public key systems embedded in the group of points on an elliptic curve over a finite field GF(p') are very appealing from a cryptologic point of view: if $g^{-1}(p')$ is the reverse cryptographic problem and $p' << p$ then $O(g_{k'}^{-1}(p')) \sim O(f_k^{-1}(p))$ i.e. the same security level is reached using key lengths **much shorter** (therefore more practical) than those in other public key systems.
- If p' = p elliptic curve cryptosystems result harder to "crack" than others because $O(g_k^{-1}(p)) >> O(f_k^{-1}(p))$.

- Elliptic curve cryptosystems involve elementary arithmetic operations that make it easy to implement (in either hardware or software).

Elliptic Curve Cryptosystems
N.Koblitz, Mathematics of Computation, (48), pp. 203-209, 1987
https://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866109-5/

Canonical Form for EC

- Generalized Weierstrass Equation of elliptic curves using affine coordinates (x,y):

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

with a_i, x, y $\in$ GF(p), p prime, or $\in$ GF(2^n), n integer

- For cryptography are of interest the following EC families:

$$y^2 = x^3 + ax + b$$ $a_1=a_2=a_3=0$, $\mathbf{a=a_4}$, $\mathbf{b=a_6}$, x, y in GF(p)

$$y^2 + xy = x^3 + ax + b$$ $a_1=1$, $a_2=a_3=0$, $\mathbf{a=a_4}$, $\mathbf{b=a_6}$, x, y in GF(2^n)

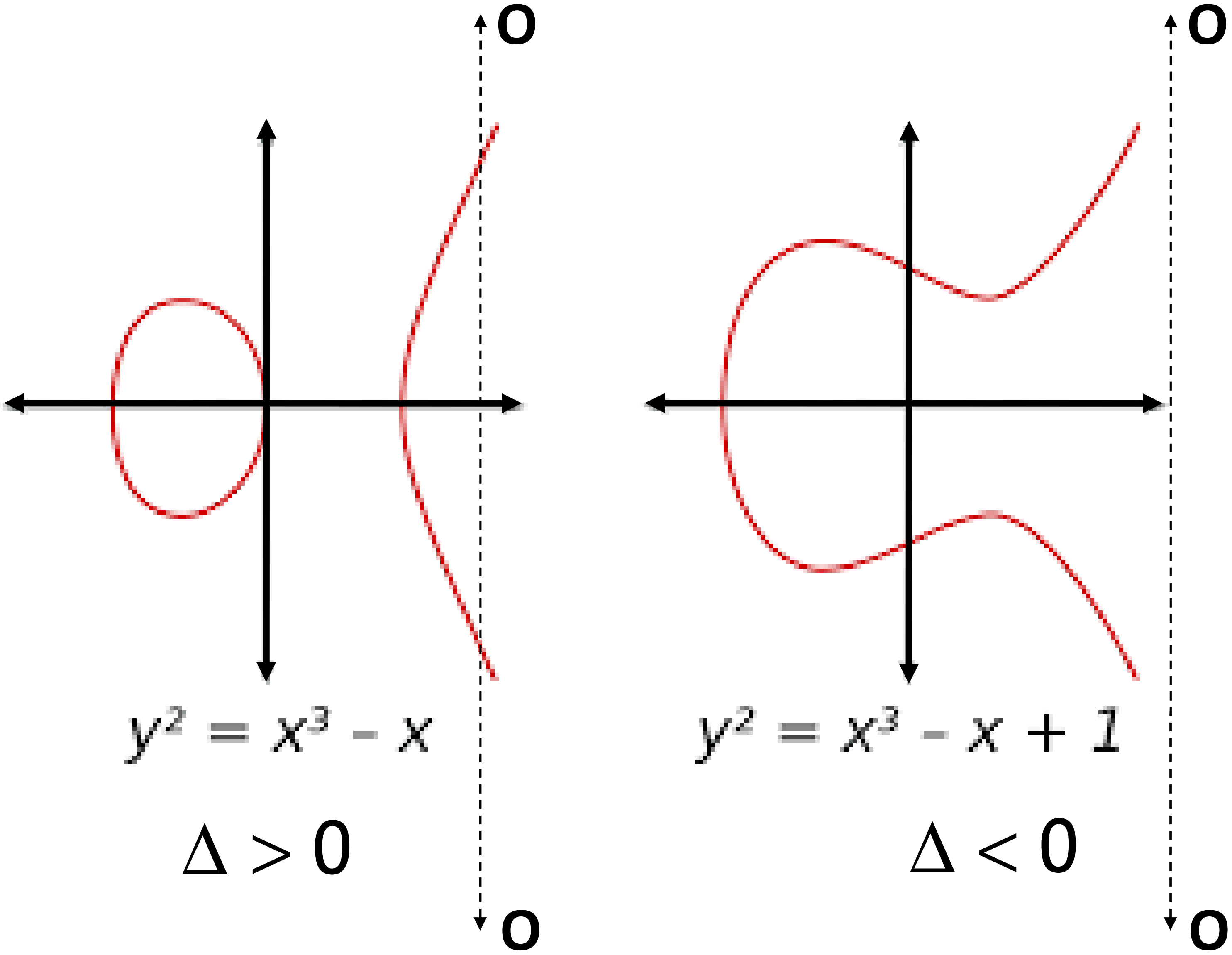
- Be $\Delta = -16(4a^3 + 27b^2) \neq 0$
- Be char(GF()) $\neq 2$ and char(GF()) $\neq 3$ (non-singular EC to avoid multiples roots)

The EC Group

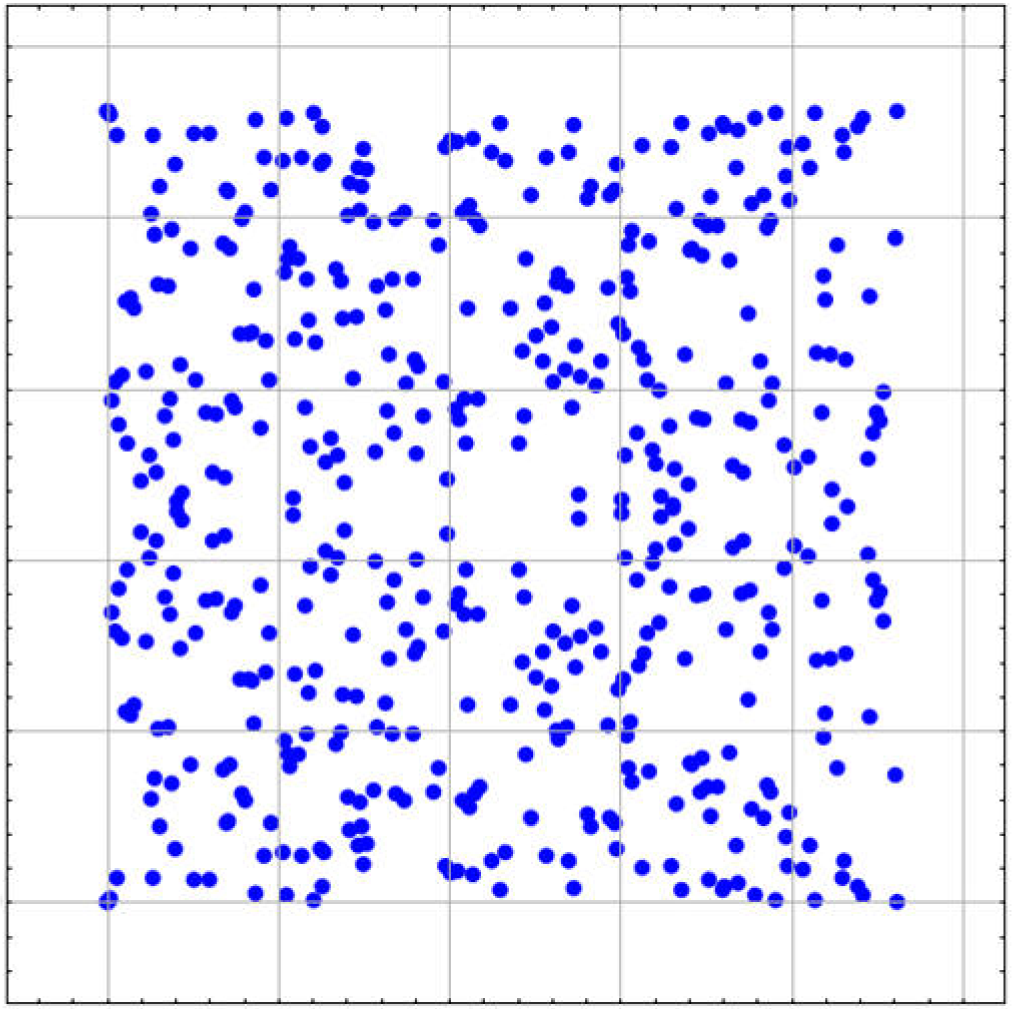
- Points of an Elliptic Curve E with coefficients in GF() or EC(GF()) constitute a finite **abelian additive cyclic group**, with the operation + ("Point Addition") and where the **O ("zero") element (identity)** is the so called **"Point at Infinity"**.
- Therefore given points P,Q,R in EC(GF()):
  - $\forall$P,Q $\in$ EC / P + Q $\in$ EC (closure)
  - (P + Q) + R = P + (Q + R) (associativity)
  - P + **O** = **O** + P = P (identity element)
  - there exists (− P) such that − P + P = P + (− P) = **O** (inverse element)
- As any cyclic group, at least one generator G (or base point) exists in EC group.
- **ord(G):** as any additive group, ord(G) = n if n is the smallest integer such that G + G + … (n times) … + G **= nG = O**.
- **EC elements (EC points)** can be generated applying iteratively Point Addition to the generator G: {G, 2G, 3G, …, (n-1)G} $\cup$ **O.**

Observation: EC are natively defined over the projective plane with homogeneous coordinates x,y,z (not over the affine plane with coordinates x/z^{α},y/z^{β}) where according to the specific values for α and β, **O is the point (*,*,0)** outside the affine plane: therefore <u>**O** is an effective point of EC</u> and in affine representations **O** must be added by construction.

Graphical Representation in R



Graphical Representation in GF



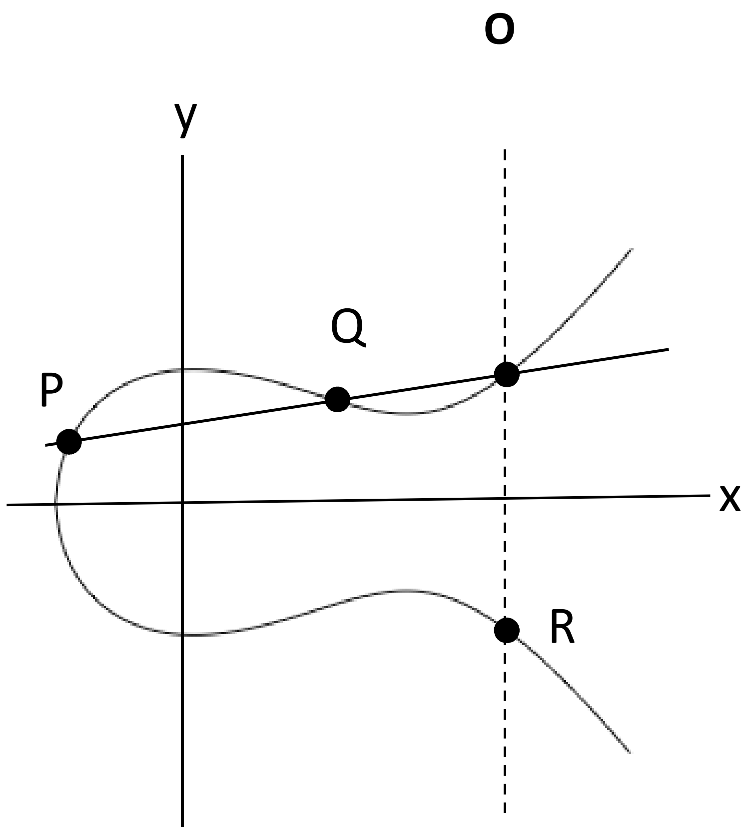
Points of a Elliptic Curve

- Consider E: $y^2 = x^3 + 2x + 3$ with coefficient in GF(5)
  $x = 0 \Rightarrow y^2 = 3 \Rightarrow$ no solution (0*0=0, 1*1=1, 2*2=4, 3*3=4, 4*4=1)
  $x = 1 \Rightarrow y^2 = 1 \Rightarrow y = 1,4$
  $x = 2 \Rightarrow y^2 = 0 \Rightarrow y = 0$
  $x = 3 \Rightarrow y^2 = 1 \Rightarrow y = 1,4$
  $x = 4 \Rightarrow y^2 = 0 \Rightarrow y = 0$

- Then points on the Elliptic Curve EC(GF(5)) are by enumeration:
  { (1,1), (1,4), (2,0), (3,1), (3,4), (4,0)} $\cup$ **O**
  Therefore the points in EC(GF(5)) are 6 plus O.
  In general the exact computation of the points in EC(GF()) is a difficult task.

- **Hasse Theorem**: order(EC(GF()) = number of points in EC(GF()) is in the range:
  $[q+1-q^{1/2}, q+1+q^{1/2}] \approx$ **q if q>>1** where $q=p^n$, p prime, n integer

Definition of Point Addition



- Consider elliptic curve
  E: $y^2 = x^3 - x + 1$ over R

- If P and Q are on E,
  the point addition

  R = P + Q

  is defined as shown in picture

- Special case of Point Addition is Point Doubling (**Q** = **P** = **G**): **R=G+G=2G**
- Iterative Point Additions is Scalar Multiplication: **R=G+G+…G=nG**
- Scalar Multiplication is energy and time consuming: these are some algebraic [tricks]to minimize computations.

EXEMERGE EXCELLENCE

Operations in GF(p) Affine Coordinates

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

$$\boxed{y^2 = x^3 + ax + b}$$

- Point Addition: R=P+Q

$$x_R = \lambda^2 - x_P - x_Q$$

$$y_R = \lambda(x_P - x_R) - y_P$$

$$\lambda = \frac{y_Q - y_P}{x_Q - x_P}$$

- Point Doubling: R=2P

$$x_R = \lambda^2 - 2x_P$$

$$y_R = \lambda(x_P - x_R) - y_P$$

$$\lambda = \frac{3x_P^2 + a}{2y_P}$$

Guide to Elliptic Curve Cryptography
D. Hankerson, A. Menezes, S. Vanstone, Ed. Spriger, ISBN 0-387-95273-X, 2004

EXEMERGE CELLENCE

Operations in $GF(2^n)$ Affine Coordinates

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

$$\boxed{y^2 + xy = x^3 + ax + b}$$

- Point Addition: R=P+Q

$$x_R = \lambda^2 + \lambda + x_P + x_Q + a$$

$$y_R = \lambda(x_P + x_R) + x_R + y_P$$

$$\lambda = \frac{y_Q + y_P}{x_Q + x_P}$$

- Point Doubling: R=2P

$$x_R = \lambda^2 + \lambda + a$$

$$y_R = x_P^2 + \lambda x_R + x_R$$

$$\lambda = x_P + \frac{y_P}{x_P}$$

Guide to Elliptic Curve Cryptography
D. Hankerson, A. Menezes, S. Vanstone, Ed. Spriger, ISBN 0-387-95273-X, 2004

Projective Coordinates

- Projective coordinates eliminate expensive **modular inversions** at the cost of cheaper modular multiplications and squares.

- Formulas in projective coordinates can be derived by first converting the points to affine coordinates, then using the formulas for Point Addition and Point Doubling to add / double the affine points, and finally clearing denominators.

- Lopez-Dahab Coordinates transformations
  - From affine to projective: $\mathbf{(x,y) \rightarrow (x,y,1)}$
  - From projective to affine: $\mathbf{(x,y,z) \rightarrow (x/z,y/z^2)}$

- Jacobi Coordinates transformations
  - From affine to projective: $\mathbf{(x,y) \rightarrow (x,y,1)}$
  - From projective to affine: $\mathbf{(x,y,z) \rightarrow (x/z^2,y/z^3)}$

Operations in GF(p) Jacobi Coordinates

$$y^2z = x^3 + axz^4 + bz^6$$

O = (0,1,0)

- Point Addition: R=P+Q

$$A = x_Q z_P^2$$
$$B = y_Q z_P^2$$
$$C = A - x_P$$
$$D = B - y_P$$
$$x_R = D^2 - (C^3 + 2x_P C^2)$$
$$y_R = D(x_P C^2 - x_R) - y_P C^3$$
$$z_R = z_P C$$

- Point Doubling: R=2P

$$A = 4x_P y_P^2$$
$$B = 8y_P^4$$
$$C = 3(x_P - z_P^2)(x_P + z_P^2)$$
$$D = -2A + C^2$$
$$x_R = D$$
$$y_R = C(A - D) - B$$
$$z_R = 2y_P z_P$$

Software implementation of the NIST elliptic curves over prime fields
M. Brown, D. Hankerson, J. Lopez Hernandez, A. Menezes, *Topics in Cryptology, CT-RSA, 2001(LNCS 2020), 250–265, 2001*

Operations in GF(2^n) Lopez-Dahab Coords

$$\boxed{y^2 + xyz = x^3z + ax^2z^2 + bz^4}$$

$\boxed{\mathbf{O} = (1,0,0)}$

- Point Addition: R=P+Q

$$A = y_Q z_P^2 + y_P$$
$$B = x_Q z_P + x_P$$
$$C = z_P B$$
$$D = B^2(C + az_P^2)$$
$$x_R = A^2 + D + AC$$
$$y_R = AC(x_R + x_Q z_R) + z_R(x_R + y_Q z_R)$$
$$z_R = C^2$$

- Point Doubling: R=2P

$$x_R = x_P^4 + bz_P^4$$
$$y_R = bz_P^4 z_R + x_R(az_R + y_P^2 + bz_P^4)$$
$$z_R = x_P^2 z_P^2$$

Software implementation of elliptic curve cryptography over binary fields
D. Hankerson, J. Lopez Hernandez, A. Menezes, *Cryptographic Hardware and Embedded Systems—CHES 2000(LNCS 1965), 1–24, 2000.*

EC Domain Parameters

- Given EC(GF(p)), p prime, the Domain Parameter associated to E is the 6-pla defined as follows
  - the prime p
  - the coefficients a and b, with a,b$\in$ GF(p)
  - the generator G
  - the order r of G
  - the co-factor h defined as #E(GF(p))/r

- Given EC(GF(2^m)), m integer, the Domain Parameter associated to E is the 6-pla defined as follows
  - the number m
  - the coefficients a and b, with a,b$\in$ GF(2^m)
  - the generator G
  - the order r of G
  - the co-factor h defined as #EC(GF(2^m))/r

The co-factor determines if the generator G and EC points refer to the group (h=1) or to a subgroup (h > 1) of order r.

Outline

- Modular Arithmetic
- Generating Prime Numbers
- Generating Pseudo-random Numbers
- Elliptic Curve Algebra
- Discrete Logarithm Problem and its EC version
- Pairings on Elliptic Curves
- Zero Knowledge Proof

Discrete Logarithm Problem and its EC version

- DLP: Given y, g in GF(p), g generator, solve:
  $y = g^x$ mod p for an integer x in [1,p-1] ($x = \log_g(y)$).
- ECDLP: Given Q, P in EC(GF(p)), P generator, solve **Q = xP mod p** for an integer **x** in [1,p-1]. The lower bound complexity for both DLP and ECDLP is (Pohlig-Hellman algorithm) is **$O(\exp(2\ln p \cdot \ln\ln p)^{1/2})$** in case of **p is a "safe prime"**
  - **$O(p^{1/2})$** in case of **p is not a "safe prime"**
- RSA: Given c, m in Z(n), n=pq, p,q primes, 2<e<n, solve **$c=m^e$ mod n** for an integer **m.** The lower bound complexity is **$O(\exp((\ln n)^{1/3} \cdot (\ln\ln n)^{2/3}))$.**

Try to replace $p = 2^{ECC\_KEY\_SIZE}$ and $n = 2^{RSA\_KEY\_SIZE}$ values listed in the NIST table into the expressions for compexity above: the same complexity is returned!

$O(\exp(2\ln p \cdot \ln\ln p)^{1/2}) \sim O(\exp((\ln n)^{1/3} \cdot (\ln\ln n)^{2/3}))$.

NIST guidelines for public key sizes for AES

ECC KEY SIZE (Bits)	RSA KEY SIZE (Bits)	KEY SIZE RATIO	AES KEY SIZE (Bits)
163	1024	1 : 6	
256	3072	1 : 12	128
384	7680	1 : 20	192
512	15 360	1 : 30	256

Supplied by NIST to ANSI X9F1

Discrete Logarithm Problem and its EC version

- The Certicom ECC Challenge:

  https://www.certicom.com/content/certicom/en/the-certicom-ecc-challenge.html
- Certicom Corp. has issued a series of ECC challenges:
  - Level I involves fields of 109-bit and 131-bit sizes.
  - Level II includes **163, 191, 239, 359-bit sizes**.

    All Level II challenges are currently believed to be computationally infeasible.
- Cryptoanalysis of cyber attacks against DLP and ECDLP requires very advanced algebraic tools out of the scope of this course.

An Improved Algorithm for Computing Logarithms over GF(p) and its Cryptographic Significance
S. Pohlig and M. Hellman, IEEE Transactions on Information Theory (24): 106–110, 1978.
Recent progress on the elliptic curve discrete logarithm problem
S. Galbraith, P. Gaudry, Designs, Codes and Cryptography, Springer Verlag, 2016, 78 (1), pp.51-72.
On the feasibility of an ECDLP algorithm
S. Grebnev, HSE Tikhonov Moscow Institute of Electronics and Mathematics (MIEM HSE), 2018
Some remarks on the elliptic curve discrete logarithm problem
Yu. Nesterenko, Mat. Vopr. Kriptogr., 2016, Vol. 7, Issue 2, 115–120
Reliability of RSA Algorithm and its Computational Complexity
M. Karpinsky, Y. Kinakh, International Scientific Journal of Computing, 2003, Vol. 2, Issue 3, 119-122

Outline

- Modular Arithmetic
- Generating Prime Numbers
- Generating Pseudo-random Numbers
- Elliptic Curve Algebra
- Discrete Logarithm Problem and its EC version
- Pairings on Elliptic Curves
- Zero Knowledge Proof

Pairing Based Cryptography

- Let G_1 and G_2 be **additive cyclic groups of order n, n prime**
- Let G_3 be a **multiplicative cyclic group of the same order n**.

The pairing ê is the map: $\boxed{\hat{\mathbf{e}}: G_1 \times G_2 \rightarrow G_3}$

- **Bilinearity**

  $\hat{\mathbf{e}}(P+P',Q)=\hat{\mathbf{e}}(P,Q)\ \hat{\mathbf{e}}(P',Q)$ for any $P,P' \in G_1$, $Q \in G_2$

  $\hat{\mathbf{e}}(P,Q+Q')=\hat{\mathbf{e}}(P,Q)\ \hat{\mathbf{e}}(P,Q')$ for any $P \in G_1$, $Q,Q' \in G_2$

- **Non-Degeneracy**

  For any non-identity point $P \in G_1$ there is a $Q \in G_2$ such that $\hat{\mathbf{e}}(P,Q) \neq 1$

  For any non-identity $Q \in G_2$ there is a $P \in G_1$ such that $\hat{\mathbf{e}}(P,Q) \neq 1$

- In case $G_2=G_1=E$ (**pairing on elliptic curves**), n is the order of the generator G of E. In this case the pairing becomes: $\boxed{\hat{\mathbf{e}}: E \times E \rightarrow G_3}$

Weil Pairing on Elliptic Curves

- Weil pairing is a mapping **from a couple of points on E order r** (pairing) over GF() **to the r-th root of unity over GF()**.
- Let $G_1 = G_2$ = EC(GF()) be an elliptic curve defined over GF().
- Let P, Q $\in$ EC(GF()) be points of order r, r prime, hence rP=O and rQ=O.
- Let μ_r be a set of r elements in GF()
- Weil Pairing $\hat{\mathbf{e}}_r$ is a bilinear, non-degenerate, alternating mapping of the form:

$$\hat{\mathbf{e}}_{\mathbf{r}} : (\mathbf{P}, \mathbf{Q}) \rightarrow \mu_{\mathbf{r}} \qquad \mu_{\mathbf{r}} = \{\mathbf{x} \in \mathbf{GF}() \mid \mathbf{x}^{\mathbf{r}} = 1\}$$

1) $\hat{\mathbf{e}}_r(\mathbf{P}_1+\mathbf{P}_2,\mathbf{Q}) = \hat{\mathbf{e}}_r(\mathbf{P}_1,\mathbf{Q})\, \hat{\mathbf{e}}_r(\mathbf{P}_2,\mathbf{Q})$ (bilinearity)

   $\hat{e}_r(\mathbf{P},\mathbf{Q}_1+\mathbf{Q}_2) = \hat{\mathbf{e}}_r(\mathbf{P},\mathbf{Q}_1)\, \hat{\mathbf{e}}_r(\mathbf{P},\mathbf{Q}_2)$ (bilinearity)

   1a) $\hat{\mathbf{e}}_r(mP,Q) = \hat{\mathbf{e}}_r(P,Q)^m = \hat{\mathbf{e}}_r(P,mQ)$, $\hat{\mathbf{e}}_r(P,nQ) = \hat{\mathbf{e}}_r(P,Q)^n = \hat{\mathbf{e}}_r(nP,Q)$,

   1b) $\hat{\mathbf{e}}_r(mP,nQ) = \hat{\mathbf{e}}_r(P,Q)^{mn} = \hat{\mathbf{e}}_r(nP,mQ)$

2) $\hat{\mathbf{e}}_r(P,Q) = 1$ for all Q iff P = O and for all P iff Q = O (non degeneracy)

3) $\hat{\mathbf{e}}_r(\mathbf{P},\mathbf{Q}) = \hat{\mathbf{e}}_r(\mathbf{Q},\mathbf{P})^{-1}$ (alternation)

   $\hat{\mathbf{e}}_r(\mathbf{P},\mathbf{P}) = \hat{\mathbf{e}}_r(\mathbf{P},\mathbf{P})^{-1}$

Sur les fonctions algébriques à corps de constantes fini

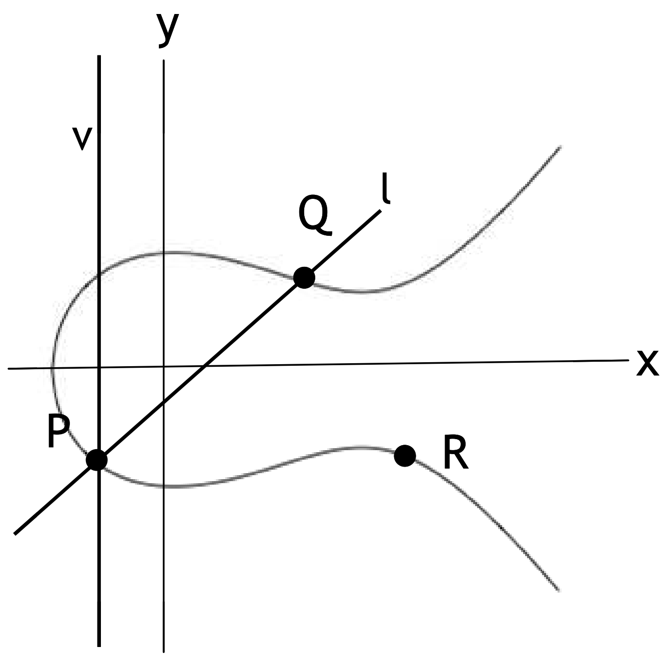
A. Weil, Les Comptes rendus de l'Académie des sciences, **210**: 592–594, MR 0002863, 1940

Weil Pairing on Elliptic Curves

ISO/IEC 14888-3:2018 recommends the costruction of a Weil Pairing e_r according Miller's algorithm:

$$e_r(P,Q) = \frac{f(Q+R)g(-R)}{f(R)g(P-R)} \qquad \mathbf{R} \notin \{\mathbf{O,P,\text{-}Q,P\text{-}Q}\}$$

- e_r is independent of choice of the functions f and g and of the point R in E
- In Miller's algorithm f is the sloped line l through P and Q and g is the vertical line v through P.



The Weil Pairing, and Its Efficient Calculation
V. S. Miller, Journal of Cryptology 17(4):235-261, 2004

Outline

- Modular Arithmetic
- Generating Prime Numbers
- Generating Pseudo-random Numbers
- Elliptic Curve Algebra
- Discrete Logarithm Problem and its EC version
- Pairings on Elliptic Curves
- Zero Knowledge Proof

Zero Knowledge Proof

Peggy's secret is (s_1=5, s_2=7, s_3=3)

p,q primes of the form 4k+3, k any integer, e.g. p=31 (4x7+3), q=23 (4x5+3)

Peggy (the prover) ⟷ n=p*q=713 ⟷ Victor (the verifier)

random r<n
e.g. r=13
x=r^2 mod n

x= 169 →

random booleans
a1, a2, a3
e.g. a1=1
a2=0
a3=1

← Challenges: a1=1, a2=0, a3=1

$y_1=r\, s_1^{a1} s_2^{a2} s_3^{a3}$

y_1= 195 →

y_1^2 = 236

$v_1=s_1^2$ mod n
$v_2=s_2^2$ mod n
$v_3=s_3^2$ mod n

v_1= 25, v_2= 49, v_3= 9 →

$y=xv_1^{a1}v_2^{a2}v_3^{a3}$ = 236
check y = y_1^2

If YES Peggy has showed to know a secret without sharing this secret.

Proof: $y_1^2 = (r\, s_1^{a1}s_2^{a2}s_3^{a3})^2 = xv_1^{a1}v_2^{a2}v_3^{a3} = y$

Zero Knowledge Proofs of Identity
U. Feige, A. Fiat, A. Shamir, Journal of Cryptology 1(2):77-94, 1988

BACKUP SLIDES

Key Equivocation

Bayes Th.

$$H(K,P,C) = H(C \mid P,K) + H(P,K) = H(P,K)$$

=0: known P and K, C = $Enc_K(P)$

Bayes Th.

$$H(K,P,C) = H(P \mid C,K) + H(C,K) = H(C,K)$$

=0: known C and K, P = $Dec_K(C)$

Thus

$$H(C,K) = H(P,K)$$

Bayes Th.

$$H(C,K) = H(K \mid C) + H(C)$$

$$H(P,K) = H(P) + H(K)$$

P and K statistically independent

$$H(K \mid C) + H(C) = H(P) + H(K)$$

$$H(K \mid C) = H(P) + H(K) - H(C)$$

q.e.d.

Key Equivocation (Lower Bound)

1st Shannon Theorem (on source coding)

$$H(P) = \lim_{n\to\infty} \frac{H(P_n)}{n} \approx \frac{H(P_n)}{n} \le \log_2|P|$$

$$H(C) = \lim_{n\to\infty} \frac{H(C_n)}{n} \approx \frac{H(C_n)}{n} \le \log_2|C|$$

P_n , C_n the random variables representing the n-gram (or n-sized bitstrings or n-blocks) of plaintext and ciphertext.

$H(P_n)$, $H(C_n)$ entropy of n-gram of plaintext and ciphertext.

H(P), H(C) entropy of the plaintext and ciphertext.

$$R_P = 1 - \frac{H(P)}{\log_2|P|} = \frac{\log_2|P| - H(P)}{\log_2|P|}$$

R_P defines the redundancy of plaintext

$$H(K \mid C_n) = H(P_n) + H(K) - H(C_n)$$

from Theorem on Key Equivocation

$$H(P_n) \approx nH(P) \qquad H(C_n) \le n\log_2|C|$$

$$H(K|C_n) \ge nH(P) + H(K) - n\log_2|C| \ge \boxed{H(K) - nR_P\log_2|P|}$$

with large n

q.e.d.

lower bound

$$H(P) = \log_2|P| - R_P\log_2|P| \qquad |P| = |C|$$

from the definition of R_P $\qquad$ $e_k()$ is an invertible function

Tricks n. 1 and n. 2

1. **Barrett Reduction:** integer modular reductions without divisions

$$a \bmod n \equiv a - \left\lfloor \frac{1}{n} a \right\rfloor n$$

2. **Square and Multiply algorithm** (to compute exponentiations)

Start p=1

Scan the exponent translated into binary from MSB to LSB

$$1 \rightarrow ()^2 a \qquad 0 \rightarrow ()^2$$

$$p = a^{19} = a^{(10011)} = (((((1)^2 a)^2)^2)^2 a)^2 a = (a^8 a)^2 a = a^{19}$$

Trick n. 3

3. Double and Add Algorithm (to compute scalar multiplication in ECC)

This is the corresponding algorithm for EC of the Square and Multiply Algorithm

Start P=O

Scan the scalar translated into binary from MSB to LSB

$$1 \rightarrow 2() + G \qquad 0 \rightarrow 2()$$

$$P = 19G = (10011)G = 2(2(2(2(2(O) + G))) + G) + G =$$

$$= 2(8G + G) + G = 19G$$

Trick n. 4

4. **Shamir's Trick:** optimizes the computation of the form aP+bQ, where a,b are integers and P,Q are two points on an elliptic curve.

   A straightforward implementation requires two scalar multiplications and a point addition.

   Shamir's trick allows to compute the above value at a cost close to one scalar multiplication.

   If the scanned bit positions are starting from MSB to LSB

   (ai = 1, bi = 0) $\rightarrow$ 2() + P

   (ai = 0, bi = 1) $\rightarrow$ 2() + Q

   (ai = 0, bi = 0) $\rightarrow$ 2()

   (ai = 1, bi = 1) $\rightarrow$ 2() + (P+Q)

Shamir Trick - Example

Example: $37P + 22Q$

$$\begin{array}{lcrrrrrr} 37 & = & 1 & 0 & 0 & 1 & 0 & 1 \\ 22 & = & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

Shamir Trick - Example

Example: $37P + 22Q$

$$\begin{array}{ccccccc} 37 & = & 1 & 0 & 0 & 1 & 0 & 1 \\ 22 & = & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

P

Shamir Trick - Example

Example: $37P + 22Q$

$$\begin{array}{rcllllll} 37 & = & 1 & 0 & 0 & 1 & 0 & 1 \\ 22 & = & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

P

2P

2P+Q

Shamir Trick - Example

Example: $37P + 22Q$

$$\begin{array}{lcllllll} 37 & = & 1 & 0 & 0 & 1 & 0 & 1 \\ 22 & = & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

P

2P

2P+Q

4P + 2Q

Shamir Trick - Example

Example: $37P + 22Q$

$$\begin{array}{lclllllll} 37 & = & & 1 & 0 & 0 & 1 & 0 & 1 \\ 22 & = & & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

P	9P+5Q
2P	
2P+Q	
4P + 2Q	
8P + 4Q	

Shamir Trick - Example

Example: $37P + 22Q$

$$
\begin{array}{lcllllll}
37 & = & 1 & 0 & 0 & 1 & 0 & 1 \\
22 & = & 0 & 1 & 0 & 1 & 1 & 0
\end{array}
$$

P	9P+5Q
2P	18P+10Q
2P+Q	18P+11Q
4P + 2Q	
8P + 4Q	

Shamir Trick - Example

Example: $37P + 22Q$

$$
\begin{array}{lcllllll}
37 & = & 1 & 0 & 0 & 1 & 0 & 1 \\
22 & = & 0 & 1 & 0 & 1 & 1 & 0
\end{array}
$$

P	9P+5Q
2P	18P+10Q
2P+Q	18P+11Q
4P + 2Q	36P + 22Q
8P + 4Q	37P + 22Q

Some definitions on Divisor Theory

- Let f be a rational function: zeros, poles, order of zeros and poles

$$f(x) = \frac{(x-1)^2}{(x+2)^3}$$

1 is a zero order (or multiplicity) 2
-2 is a pole order 3
∞ is a zero order 1

- The **divisor** of f is **div(f)** =2(1)+1(∞)-3(-2)
- **Degree of a divisor** deg(div(f)) is defined as the sum of the orders of zeros and poles of f. In the example the degree is 2+1-3=0
- Divisors D_1 and D_2 are **equivalent** or $\mathbf{D_1 \sim D_2}$ if $D_1=D_2+\text{div}(f)$ for some f.
- **Support of a divisor** D is **supp(D)**=$\{P\in E \,/\, n_P \neq 0\}$.
- D_1 and D_2 have **disjoint support** if $\text{supp}(D_1) \cap \text{supp}(D_2) = \varnothing$.
- Let f and g be rational functions defined on some field F. If div(f) and div(g) have disjoint support, then f(div(g))=g(div(f)) (Weil reciprocity).
- Let E be an elliptic curve. <u>For any function f on E is deg(div(f)) = 0</u> (theorem).

  Let $P, Q \in E(F_{q^k})$ and let D_P and D_Q be **degree zero divisors** with **disjoint supports** such that $\mathbf{D_P \sim (P) - (O)}$ and $\mathbf{D_Q \sim (Q) - (O)}$.

  There exist functions f and g such that **(f) = r$\mathbf{D_P}$** and **(g) = r$\mathbf{D_Q}$**.

  The Weil Pairing is defined by: $$e_r(P,Q) = \frac{f(D_q)}{g(D_P)}$$

Computing Weil Pairing

Miller's algorithm to evaluate e_n = < P,Q >$_n$

1. Given P,Q with order n, choose R with order n and $R \neq \infty, P, -Q, P-Q$.
2. Write n in binary as $n = (n_t, \ldots, n_1, n_0)$.
3. Set $f = 1$, $T = P$ and $i = t$.
4. If $i < 0$ then go to step 5. Else do the following:

 (a) Let l be the tangent line to E through T. Let v be the vertical line through $2T$.

 (b) Set $T = 2T$.

 (c) Set $f = f^2 \dfrac{l(Q+R)v(R)}{v(Q+R)l(R)}$

 (d) If $n_i = 1$ then do the following:

  i. Let l be the line through T and P, and v the vertical line through $T + P$.

  ii. Set $T = T + P$.

  iii. Set $f = f \dfrac{l(Q+R)v(R)}{v(Q+R)l(R)}$

 (e) Set $i = i - 1$ and return to step 4

5. The desired value is < P,Q >$_n$ = f.

The Weil Pairing, and Its Efficient Calculation
V. S. Miller
J. Cryptology, vol. 17, pp. 235-261, 2004
pages.cs.wisc.edu/~cs812-1/miller04.pdf

RSA Generator - Algorithm

- based on the RSA one-way function:
  - $x_i = x_{i-1}^b \bmod n$ $\quad\quad\quad\quad$ $i \geq 1$

  where
  - x_0 is the seed
  - $n = p*q$, p and q are large primes
  - b s.t. gcd $(b, \phi(n)) = 1$ where $\phi(n) = (p-1)(q-1)$
  - **n and b are public, p and q are secret**

  Output

  $(x_1, x_2, \ldots, x_k)$

  $y_i = x_i \bmod 2$

  $Y = (y_1 y_2 \ldots y_k)$ ← pseudo-random sequence of K bits

  Euler's Generalization of Fermat's Little Theorem:

  If gcd(a,n)=1 then $a^{\phi(n)} \bmod n = 1$ where $\phi(n)$={#a<n s.t. gcd(a,n)=1}

Blum-Micali Generator - Algorithm

- based on the discrete logarithm one-way function:
  - let p be a prime then Z_p is a cyclic group
  - let x_0 be a seed

$x_i = g^{x_{i-1}} \bmod p$ $\qquad\qquad$ $i \geq 1$

Output

$(x_1, x_2, \ldots, x_k)$

$y_i = 1$ if $x_i \geq (p-1)/2$

$y_i = 0$ otherwise

$Y = (y_1 y_2 \ldots y_k)$ ← pseudo-random sequence of K bits

Blum-Blum-Shub Generator - Algorithm

- based on the squaring one-way function
  - Let p, q be primes with p≡q=3 mod 4
  - Let n = p*q
  - Let x_0 be a seed

$$x_i = x_{i-1}^2 \mod n \qquad i \geq 1$$

Output

$(x_1, x_2, \dots, x_k)$

$y_i = x_i \mod 2$

$Y = (y_1 y_2 \dots y_k)$ ← pseudo-random sequence of K bits