Authenticated Key Agreement Protocol using ElGamal Encryption and Signature.
Hybrid encryption for a large files combining asymmetric and symmetric encryption method.

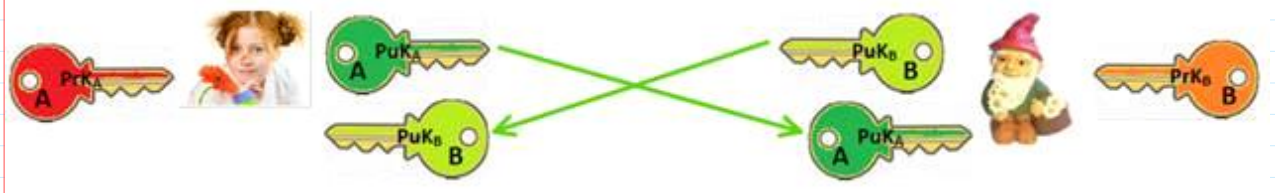
Hybrid encryption. Let ***M*** be a large finite length file, e.g. of gigabytes length.
Then to encrypt this file using asymmetric encryption is extremely ineffective since we must split it into millions of parts having 2048 bit length and encrypt every part separately.
The solution can be found by using **asymmetric encryption** together with **symmetric encryption**, say **AES-128**.
It is named as **hybrid encryption method**.
For this purpose the **Key Agreement Protocol** (**KAP**) using **asymmetric encryption** for the same symmetric secret key ***k*** agreement must be realized and encryption of ***M*** realized by **symmetric encryption** method, say **AES-128**.

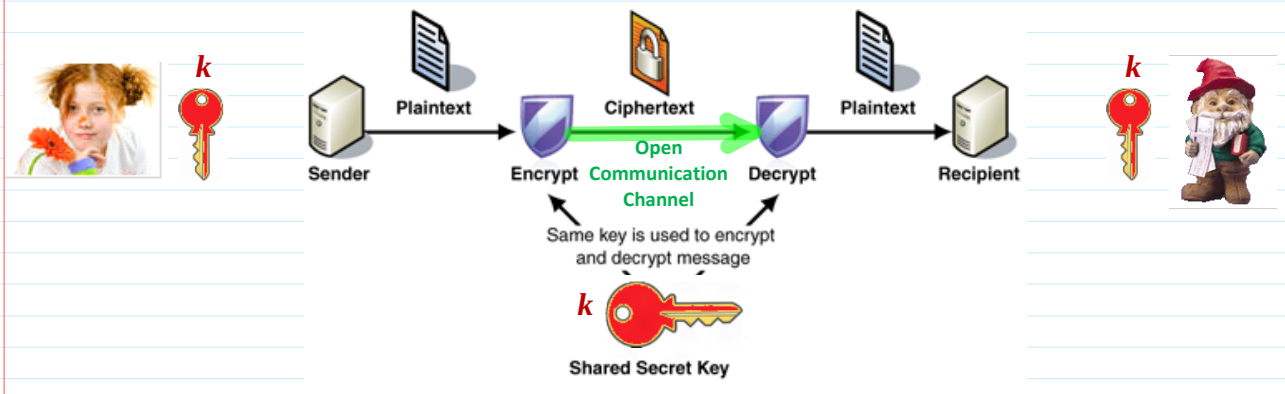
AKAP: Asym.Enc & Digital Sign.

How to encrypt large data file M: Hybrid enc-dec method.
1. Parties must agree on common symmetric secret key k.
for symmetric block cipher, e.g. AES-128, 192, 256 bits.

A: $PrK_A = x$; $PuK_A = a$.
$PuK_B = b$.

B: $PrK_B = y$; $PuK_B = b$.
$PuK_A = a$.





To
1) $k \leftarrow randi(2^{128})$
$i_k \leftarrow randi(2^{128})$

$Enc(PuK_B = b, i_k, k) = c = (E, D)$

1.1. Verify if PuK_A and $Cert_A$ are valid?
1.2. Verify if $S_{PuK_A}(h) = H(A)$ is valid?

$Enc(PuK_B = b, i_k, k) = c = (E, D)$

2) M - large file to be encrypted

$E_k(M) = AES_k(M) = G$

3) Signs ciphertext G

3.1) $h = H(G)$

3.2) $Sign(PrK_A = x, h) = \sigma = (r, s)$

c, G
σ, PuK_A
$Cert_A$

1.1. Verify if PuK_A and $Cert_A$ are valid?

1.2. Verify if σ on $h = H(G)$ is valid?

$h' = H(G)$

$Ver(PuK_A, \sigma, h') = True$

2. $Dec(PrK_B, c) = k$

3. $D_k(G) = AES_k(G) = M.$

A was using so called encrypt-and-sign (E-&-S) paradigm. (E-&-S) paradigm is recomended to prevent so called choosen ciphertext Attacks - CCA: it is most strong attack but most complex in realization.

Cookery recipe

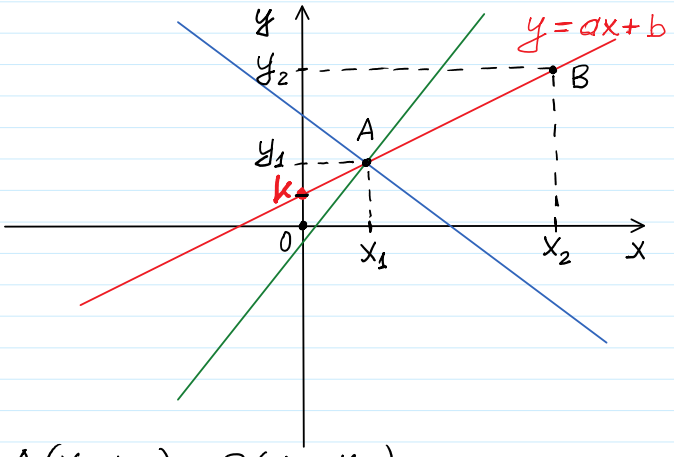
Secret Sharing scheme

Shamir's Secret Sharing (SSS) is used to secure a secret in a distributed way, most often to secure other encryption keys. The secret is split into multiple parts, called **shares**. These shares are used to reconstruct the original secret.
To unlock the secret via Shamir's secret sharing, a minimum number of shares are needed. This is called the **threshold**, and is used to denote the minimum number of shares needed to unlock the secret. An adversary who discovers any number of shares less than the threshold will not have any additional information about the secured secret-- this is called perfect secrecy. In this sense, SSS is a generalisation of the one-time pad - Vernam cipher (which is effectively SSS with a two-share threshold and two shares in total).
Let us walk through an example:
Problem: Company XYZ needs to secure their vault's passcode. They could use something standard, such as AES, but what if the holder of the key is unavailable or dies? What if the key is compromised via a malicious hacker or the holder of the key turns rogue, and uses their power over the vault to their benefit?
This is where SSS comes in. It can be used to encrypt the vault's passcode and generate a certain number of shares, where a certain number of shares can be allocated to each executive within Company XYZ. Now, only if they pool their shares can they unlock the vault. The threshold can be appropriately set for the number of executives, so the vault is always able to be accessed by the authorized individuals. Should a share or two fall into the wrong hands, they couldn't open the passcode unless the other executives cooperated.
From <https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing>

Shamir's Secret Sharing is an ideal and perfect (***t***, ***N***)-**threshold** scheme.
In such a scheme, the aim is to divide a secret ***k*** which is a ***secret key*** to decrypt a receipt
Is divided into ***N*** pieces of data ***P1***, ***P2***, ..., ***PN*** known as **shares** in such a way that:

1. Knowledge of any ***t*** or more P_i pieces makes ***k*** easily computable. Therefore ***t*** is is named as **threshold** . That is, the complete secret ***k*** can be reconstructed from any combination of ***t*** or more pieces of data.
2. Knowledge of any ***t***-1 or fewer P_i pieces leaves ***k*** completely undetermined, in the sense that the possible values for ***k*** seem as likely as with knowledge of **0** pieces. The secret ***k*** cannot be reconstructed with fewer than ***t*** pieces.
   If ***t=N***, then every piece of the original secret is required to reconstruct the secret.

We are considering the field of real numbers $\langle R, +, -, *, : \rangle$.
Then the plain consisting of real numbers is $R^2 = \{(x, y);\ x \in R, y \in R\}$



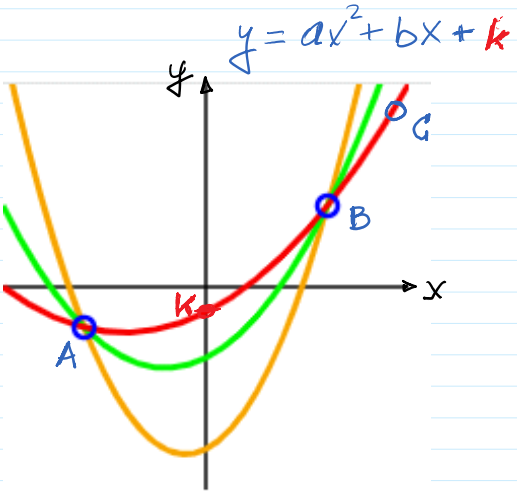
$A(x_1, y_1);\ B(x_2, y_2).$

$$\begin{cases} ax_1 + b = y_1 \\ ax_2 + b = y_2 \end{cases}$$

$$a(x_1 - x_2) = y_1 - y_2 \Rightarrow a = \frac{y_1 - y_2}{x_1 - x_2}$$

$$b = y(x = 0) = (ax + b)\Big|_{x=0}$$

$$b = k$$



One can draw an infinite number of polynomials of degree 2 through 2 points. 3 points are required to define a unique polynomial of degree 2. This image is for illustration purposes only — Shamir's scheme uses polynomials over a finite field.

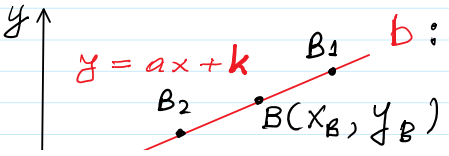
From <https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing>

$$y = ax^2 + bx + k$$

$A(x_1, y_1);\ B(x_2, y_2);\ C(x_3, y_3)$

By solving this linear system of equation, parabola coefficients a, b, k can be obtained. $\Leftarrow$

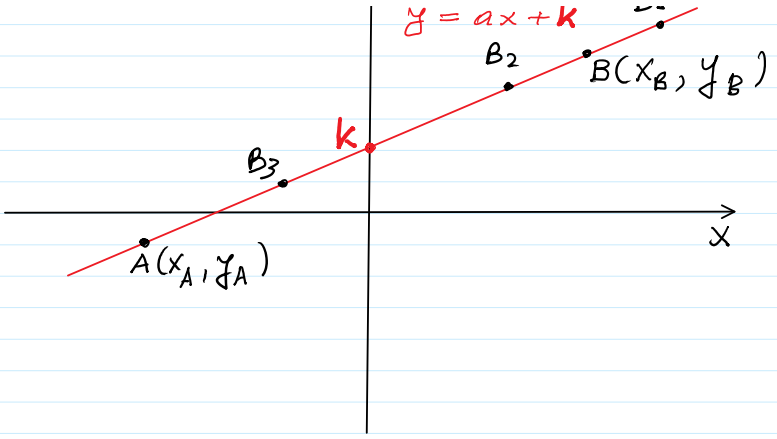
$$\begin{cases} ax_1^2 + bx_1 + c = y_1 \\ ax_2^2 + bx_2 + c = y_2 \\ ax_3^2 + bx_3 + c = y_3 \end{cases}$$



b :

$E_k(Rec) = C$; $Rec = D_k(C)$

Rec : secret recipe



Rec : secret recipe

$E(k, Rec) = C_{Rec}$

$D(k, C_{Rec}) = Rec$

shares : $\{A, B, B_1, B_2, B_3\}$

In the case of linear interpolation Threshold = 2

If (A, B) can not participate in recovery secret k, then secret k can be recovered by any other pair $(B_1, B_2), (B_1, B_3), (B_2, B_3)$

In general, secret k can be recovered by C_5^2 pairs

$(A, B), (A, B_1), (A, B_2), \ldots, (B_2, B_3)$ $\quad C_5^2 = \frac{5 \cdot 4}{2} = 10$

But 2-shares could be not enough to protect the secret k. due to bribering ....

Let it be 3-share created to protect the secret.

It is required to choose parabola $y = ax^2 + bx + k$

which can be recovered by Lagrangian interpolation using 3-points

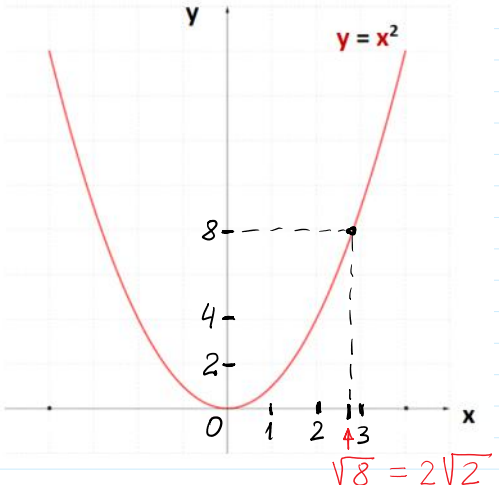
$(A, B, C) \Rightarrow$ threshold $= 3$

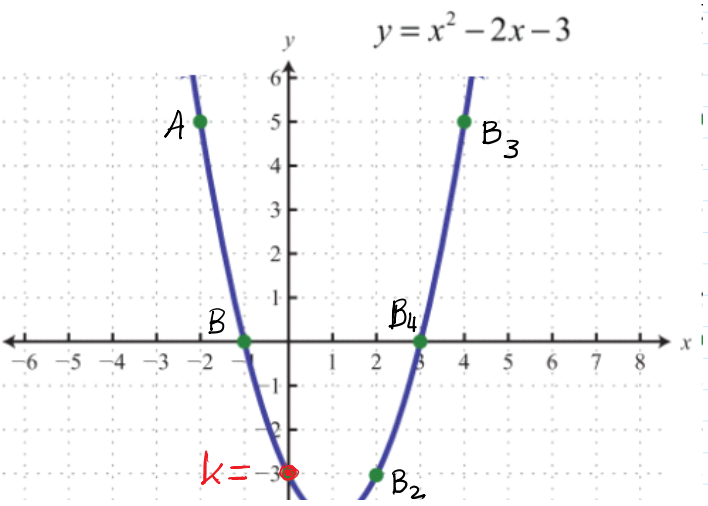
A decided to share the secret to 6-parts among

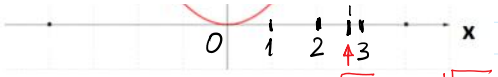
$\{A, B, B_1, B_2, B_3, B_4\}$

The number of triplets to recover secret k is

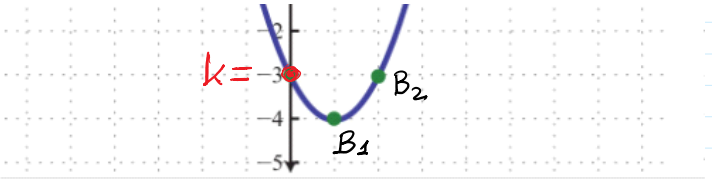
$C_6^3 = \frac{\cancel{6} \cdot 5 \cdot 4}{\cancel{3 \cdot 2 \cdot 1}} = 20.$







$\sqrt{8} = 2\sqrt{2}$



$$\begin{cases} a x_1^2 + b x_1 + k = y_1 \\ a x_2^2 + b x_2 + k = y_2 \\ a x_3^2 + b x_3 + k = y_3 \end{cases} \quad t = 3$$

$n = t - 1 = 3 - 1 = 2.$

$y = a x^2 + b x + k$

$k = -3$

For any $t = n+1$ we must choose n-th degree polynomial

$$y = P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0; \quad a_0 = k.$$

This polynomial can be recovered by Lagrange interpolation technique having $n+1$ points - uniquely recovered!

$\{P_1, P_2, \dots, P_{n+1}\} \leftarrow \{(x_1, y_1), (x_2, y_2), \dots, (x_{n+1}, y_{n+1})\}$

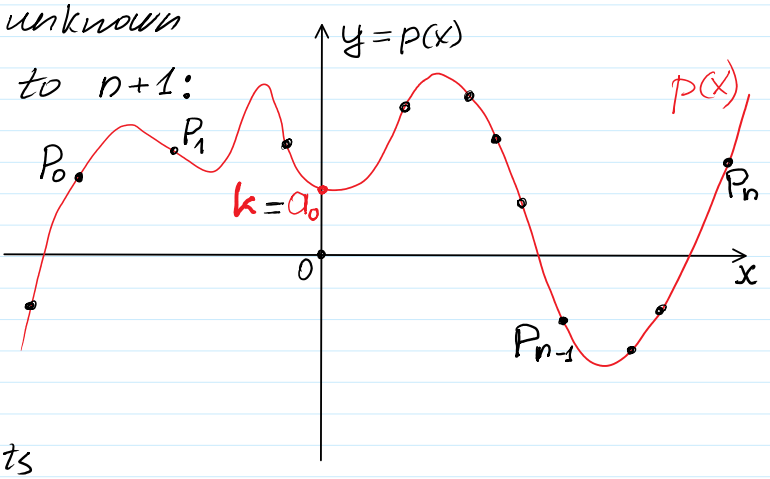
$$\begin{cases} a_n x_1^n + a_{n-1} x_1^{n-1} + \dots + a_0 = y_1 \\ \dots \\ a_n x_{n+1}^n + a_{n-1} x_{n+1}^{n-1} + \dots + a_0 = y_{n+1} \end{cases} \Rightarrow (a_n, a_{n-1}, \dots, a_1, a_0)$$

$a_0 \rightarrow k$

Let $p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ be a polynomial of order n.

In $p(x)$ the number of unknown coefficients is equal to $n+1$: to define $p(x)$ it is required to construct $n+1$ linear equations to find coefficients $\{a_0, a_1, \dots, a_{n-1}, a_n\}$.

We must have $(n+1)$ points $\{P_0, P_1, \dots, P_{n-1}, P_n\}$ where $p(x)$ is crossing these poits.



This technique is named Lagrangian interpolation: $t - 1 = n$.

$$L_{t-1}(x) = \sum_{i=1}^{t-1} y_i \prod_{j=1, j \neq i}^{t-1} \frac{x - x_j}{x_i}$$

This technique is named Lagrangian interpolation. $i = 1, \ldots$

$$k = a_0 = p(x=0) = \sum_{i=0}^{t-1} y_i \prod_{\substack{j=0 \\ i \neq j}}^{t-1} \frac{x_j}{x_j - x_i}$$

Infinite field R must be replaced by finite field $F_p = \mathcal{L}_p$.

Arithmetic of Finite fields $\mathbf{Z_p} = \mathbf{F_p} = \{0, 1, 2, \ldots, \mathbf{p}\text{-}1\}$, when **p** is prime.

$+ \bmod p$, $- \bmod p$, $* \bmod p$, $: \bmod p$: mod p by "0": $z/0$ - is not defined.

1) $\mathcal{L}_p$ in an additive group: $< \mathcal{L}_p, + \bmod p >$
2) $\mathcal{L}_p$ has multiplicative group $\mathcal{L}_p^* = \{1, 2, 3, \ldots, p-1\}$
$\mathcal{L}_p^* \subset \mathcal{L}_p$ $< \mathcal{L}_p^*, * \bmod p >$
3) The distributive law takes place in $\mathcal{L}_p$:
for all $a, b, c \in \mathcal{L}_p$: $a*(b+c) = (a*b + a*c) \bmod p$

$$\prod_{i=1}^{3} a_i = a_1 \circ a_2 \circ a_3$$

$\mathcal{L}_{11} = \{0, 1, 2, \ldots, 10\}$: $+, -, \bullet, :$ mod 11

DSA - Digital Signature Algorithm - the principle

So far we considered a group $\mathcal{L}_p^* = \{1, 2, 3, \ldots, p-1\}$
Part of this group elemets are a generators ($\sim 40\%$) and other elements are not.

Discrete Exponent Function (8/14)

$p = 11$

The results of any binary operation (multiplication, addition, etc.) defined in any finite group is named *Cayley table* including multiplication table, addition table etc.

Multiplication table of multiplicative group $\mathbf{Z_{11}^*}$ is represented below.

Multiplication tab. mod 11	$\mathbf{Z_{11}^*}$									
*	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	1	3	5	7	9
3	3	6	9	1	4	7	10	2	5	8
4	4	8	1	5	9	2	6	10	3	7
5	5	10	4	9	3	8	2	7	1	6
6	6	1	7	2	8	3	9	4	10	5
7	7	3	10	6	2	9	5	1	8	4
8	8	5	2	10	7	4	1	9	6	3

Values of inverse elements in $\mathbf{Z_{11}^*}$

$1^{-1} = 1 \bmod 11$
$2^{-1} = 6 \bmod 11$
$3^{-1} = 4 \bmod 11$
$4^{-1} = 3 \bmod 11$
$5^{-1} = 9 \bmod 11$
$6^{-1} = 2 \bmod 11$
$7^{-1} = 8 \bmod 11$
$8^{-1} = 7 \bmod 11$

9	9	7	5	3	1	10	8	6	4	2
10	10	9	8	7	6	5	4	3	2	1

$9^{-1}= 5 \bmod 11$
$10^{-1}= 10 \bmod 11$

Discrete Exponent Function (9/14)

The table of exponent values for $p = 11$ in Z_{11}^* computed **mod** 11 and is presented in table below.
Notice that according to Fermat little theorem for all $z \in Z_{11}^*$, $z^{p-1} = z^{10} = z^0 = 1$ **mod** 11.

Exponent tab. mod 11	Z_{11}^*										
^	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	*10*
1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	5	10	9	7	3	6	1
3	1	3	9	5	4	1	3	9	5	4	1
4	1	4	5	9	3	1	4	5	9	3	1
5	1	5	3	4	9	1	5	3	4	9	1
6	1	6	3	7	9	10	5	8	4	2	1
7	1	7	5	2	3	10	4	6	9	8	1
8	1	8	9	6	4	10	3	2	5	7	1
9	1	9	4	3	5	1	9	4	3	5	1
10	1	10	1	10	1	10	1	10	1	10	1

$2^2 \neq 1$ **mod** 11 & $2^5 \neq 1$ **mod** 11
$6^2 \neq 1$ **mod** 11 & $6^5 \neq 1$ **mod** 11
$7^2 \neq 1$ **mod** 11 & $7^5 \neq 1$ **mod** 11
$8^2 \neq 1$ **mod** 11 & $8^5 \neq 1$ **mod** 11

Discrete Exponent Function (10/14)

Notice that there are elements satisfying the following different relations, for example:

$$3^5 = 1 \textbf{ mod } 11 \text{ and } 3^2 \neq 1 \textbf{ mod } 11.$$

The set of such elements forms a subgroup of prime order $q = 5$ if we add to these elements the *neutral group element* 1.

This subgroup has a great importance in cryptography we denote by

$$G_5 = \{1, 3, 4, 5, 9\}.$$

The multiplication table of G_5 elements extracted from multiplication table of Z_{11}^* is presented below.

Multiplication tab. mod 11	*G*5				
*	**1**	**3**	**4**	**5**	**9**
1	1	3	4	5	9
3	3	9	1	4	5
4	4	1	5	9	3
5	5	4	9	3	1
9	9	5	3	1	4

Values of inverse elements in G_5

$1^{-1}= 1 \bmod 11$
$3^{-1}= 4 \bmod 11$
$4^{-1}= 3 \bmod 11$
$5^{-1}= 9 \bmod 11$
$9^{-1}= 5 \bmod 11$

Exponent tab. mod 11	*G*5					
^	**0**	**1**	**2**	**3**	**4**	**5**
1	1	1	1	1	1	1
3	1	3	9	5	4	1
4	1	4	5	9	3	1
5	1	5	3	4	9	1
9	1	9	4	3	5	1

Let p - is strong prime : $p \sim 2^{2048}$

$p = 2 \cdot q + 1$; e.g. if $p = 11 \rightarrow p = 2 \cdot 5 + 1 \rightarrow q = 5$ q - Is prime

$\mathcal{I}_{11}^* = \{1, 2, 3, \ldots, 10\}$; $G_2 = \{1, 10\}$; $G_5 = \{1, 3, 4, 5, 9\}$

$|\mathcal{I}_{11}^*| = 10 = 2 \cdot 5$; $|G_2| = 2$; $|G_5| = 5$;

$Ord(G_2) = 2$ & $Ord(G_5) = 5$ are prime orders.

T. If cyclic group has prime order, then all its elements except

neutral element 1 are generators.

Let p is strong prime, i.e. $p = 2 \cdot q + 1$. q - ìs prime

Then $\mathbb{Z}_p^*$ contains a subgroup G_q where all elements except 1 are generators.

Let $p \sim 2^{2048} \rightarrow |\mathbb{Z}_p^*| = p - 1$

$|G_q| = p - 1 / 2 \quad < G_q, * \bmod p >$

Taking a look at Exponent tables of $\boldsymbol{Z_p^*}$ and $\boldsymbol{Z_q}$ it is seen that the properties of generators are different.

1. The element g in $\boldsymbol{Z_p^*}$ is a generator if and only if $g^2 \neq 1$ **mod** p & $g^q \neq 1$ **mod** p.
2. The element γ in $\boldsymbol{Z_q}$ is a generator if and only if $\gamma^2 \neq 1$ **mod** p & $\gamma^q = 1$ **mod** p.

The generators in both sets can be found by choosing (randomly) a candidate element z in $\boldsymbol{Z_p^*}$ and vrifying Conditions either (1) or (2).

El-Gamal Signature Scheme **(ElG-Sig).**

Schnorr Signature Scheme **(S-Sig).**

In **Digital Signature Algorithm - DSA** the group G_q is used instead of $\boldsymbol{Z_p^*}$, since it significantly increases the security against cryptanalytic attacks due to the fact that all elements (except 1) are generators in $\boldsymbol{Z_q}$.
Group $\boldsymbol{Z_q}$ is also used in other cryptographic methods based on DEF.

Elliptic Cuve Digital Signature Algorithm -ECDSA

ECDSA	**ElGamal Signature**
EC point group: secp256k1 $y=x^3+ax+b \bmod p$; a, b, x, y is in finite field $\mathbf{F}_p = \{0, 1, 2, ..., p-1\}$; p- is of order 2^{256} G-generator point in EC	$\mathbf{Z}_p^* = \{1, 2, ..., p-1\}$, $p \sim 2^{2048}$ g-generator
$\boldsymbol{h} = H(\boldsymbol{m})$;	$\boldsymbol{h} = H(\boldsymbol{m})$;
$\boldsymbol{i} \leftarrow$randi; Compute $\boldsymbol{i}^{-1} \bmod p$	$\boldsymbol{i} \leftarrow$randi; gcd($\boldsymbol{i}$, p-1)=1 Compute $\boldsymbol{i}^{-1} \bmod (p-1)$
$\boldsymbol{R} = \boldsymbol{i} * G = \boldsymbol{i} * (x_G, y_G) = (x_R, y_R)$; $\boldsymbol{r} = x_R \bmod p$; $\lvert\boldsymbol{i}\rvert \leq 256$ bits;	$\boldsymbol{r} = g^{\boldsymbol{i}} \bmod p$;
$\boldsymbol{s} = (\boldsymbol{h} + z \bullet \boldsymbol{r}) \boldsymbol{i}^{-1} \bmod p$; $\lvert\boldsymbol{s}\rvert \leq 256$ bits;	$\boldsymbol{s} = (\boldsymbol{h} - x \bullet \boldsymbol{r}) \boldsymbol{i}^{-1} \bmod (p-1)$;
$\boldsymbol{s}^{-1} = (\boldsymbol{h} + z \bullet \boldsymbol{r})^{-1} \boldsymbol{i} \bmod p$;	$\mathbf{h} = x\boldsymbol{r} + \boldsymbol{i}\boldsymbol{s} \bmod (p-1)$.
Sign($\mathbf{PrK}_{ECC}$=z, $\boldsymbol{h}$) = ($\boldsymbol{r}$, $\boldsymbol{s}$) = $\boldsymbol{\sigma}$;	**Sign**(**PrK**=x, $\boldsymbol{h}$) = ($\boldsymbol{r}$, $\boldsymbol{s}$) = $\boldsymbol{\sigma}$;
ECDSA Verification	**ElGamal Signature Verification**
Compute $\boldsymbol{u}_1 = \boldsymbol{h} \bullet \boldsymbol{s}^{-1} \bmod p$ and $\boldsymbol{u}_2 = \boldsymbol{r} \bullet \boldsymbol{s}^{-1} \bmod p$;	Compute: $\boldsymbol{u}_1 = g^{\boldsymbol{h}} \bmod p$; and $\boldsymbol{u}_2 = a^{\boldsymbol{r}} \boldsymbol{r}^{\boldsymbol{s}} \bmod p$
Compute $\boldsymbol{R} = \boldsymbol{u}_1 * G + \boldsymbol{u}_2 * A = (x_R, y_R)$;	Signature is valid if: $\boldsymbol{u}_1 = \boldsymbol{u}_2$
The signature is valid if $\boldsymbol{r} = x_R \bmod p$.	

Schnorr Signature
$\mathbf{Z}_p^* = \{1, 2, ..., p-1\}$, $p \sim 2^{2048}$ g-generator
$\boldsymbol{h} = H(\boldsymbol{m})$;
$\boldsymbol{i} \leftarrow$randi;
$\boldsymbol{r} = g^{\boldsymbol{i}} \bmod p$;
$\boldsymbol{s} = (\boldsymbol{i} + x \bullet \boldsymbol{h}) \bmod (p-1)$;
Sign(**PrK**=x, $\boldsymbol{h}$) = ($\boldsymbol{r}$, $\boldsymbol{s}$) = $\boldsymbol{\sigma}$;
Schnorr Signature Verification
Compute: $\boldsymbol{u}_1 = g^{\boldsymbol{s}} \bmod p$. and $\boldsymbol{u}_2 = \boldsymbol{r} a^{\boldsymbol{h}} \bmod p$
Signature is valid if: $\boldsymbol{u}_1 = \boldsymbol{u}_2$

Schnorr Signature Scheme **(S-Sig).**

In general, to create a signature on the message of any finite length M parties are using cryptographic secure H-function (message digest).
In Octave we use H-function
>> hd28('…') % the input '…' of this function represents a string of symbols between the commas.
% the output of this function is decimal number having at most 28 bits.

Let M be a message in string format to be signed by **Alice** and sent to **Bob**: >> M='Hello Bob'
For signature creation **Alice** uses public parameters **PP=(*p*, *g*)** and
Alice's key pair is $PrK_A=x$, $PuK_A= a = g^x \bmod p$.

Alice chooses at random u, $1<u<p-1$ and computes first component r of his signature:

$$r=g^u \bmod p. \qquad (2.19)$$

Alice computes H-function value h and second component s of her signature:

$$h=H(M||r), \qquad (2.20)$$

$$s=u+xh \bmod (p-1). \qquad (2.21)$$

Alice's signature on h is $\sigma=(r, s)$. Then **Alice** sends M and σ to **Bob**.

After receiving M' and σ, Bob according to (2.20) computes h'

$$h'=H(M'||r),$$

and verifies if

$$g^s \bmod p = ra^{h'} \bmod p. \qquad (2.22)$$

V1 V2

Symbolically this verification function we denote by

$$Ver(a,\sigma,h')=V\in\{True, False\}\equiv\{1, 0\}. \qquad (2.23)$$

This function yields *True* if (2.22) is valid if: $h=h'$ and $PuK_A= a =F(PrK_A)= g^x \bmod p$.
and: $M=M'$

Discrete Exponent Function (11/14)

Notice that since G_5 is a subgroup of Z_{11}^* the multiplication operations in it are performed **mod** 11.
The exponent table shows that all elements {3, 4, 5, 9} are the generators in G_5.
Notice also that for all $\gamma\in\{3, 4, 5, 9\}$ their exponents 0 and 5 yields the same result, i.e.

$$\gamma^0 = \gamma^5 = 1 \bmod 11.$$

This means that exponents of generators γ are computed **mod** 5.

This property makes the usage of modular groups of prime order q valuable in cryptography since they provide a higher-level security based on the stronger assumptions we will mention later.

Therefore, in many cases instead the group Z_p^* defined by the prime (not necessarily strong prime) number p the subgroup of prime order G_q in Z_p^* is used.

In this case if p is strong prime, then generator γ in G_q can be found by random search satisfying the following conditions

$$\gamma^q = 1 \bmod p \text{ and } \gamma^2 \neq 1 \bmod p.$$

Analogously in this generalized case this means that exponents of generators γ are computed **mod** q. In our modeling we will use group Z_p^* instead of G_q for simplicity.

Discrete Exponent Function (12/14)

Let as above p=11 and is strong prime and generator we choose g = 7 from the set Γ={2, 6, 7, 8}.
Public Parameters are **PP**=(11,7), Then $\mathbf{DEF}_g(x) = \mathbf{DEF}_7(x)$ is defined in the following way:

$$\mathbf{DEF}_7(x) = 7^x \bmod 11 = a;$$

$\mathbf{DEF}_7(x)$ provides the following 1-to-1 mapping, displayed in the table below.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
7^x **mod** p = a	1	7	5	2	3	10	4	6	9	8	1	7	5	2	3

You can see that a values are repeating when x = 10, 11, 12, 13, 14, etc. since exponents are reduced **mod** 10 due to *Fermat little theorem*.

The illustration why 7^x **mod** p values are repeating when x = 10, 11, 12, 13, 14, etc. is presented in computations below:

10 mod 10 = 0; 710 = 70 = 1 mod 11 = 1.
11 mod 10 = 1; 711 = 71 = 7 mod 11 = 7.
12 mod 10 = 2; 712 = 72 = 49 mod 11 = 5.
13 mod 10 = 3; 713 = 73 = 343 mod 11 = 2.
14 mod 10 = 4; 714 = 74 = 2401 mod 11 = 3.
etc.

Discrete Exponent Function (13/14)

For illustration of 1-to-1 mapping of $\mathbf{DEF}_7(x)$ we perform the following step-by-step computations.

	$x \in Z_{10}$	$a \in Z_{11}^*$
$7^0 = 1$ **mod** 11	**0**	**1**
$7^1 = 7$ **mod** 11	**1**	**2**
$7^2 = 5$ **mod** 11	**2**	**3**
$7^3 = 2$ **mod** 11	**3**	**4**
$7^4 = 3$ **mod** 11	**4**	**5**
$7^5 = 10$ **mod** 11	**5**	**6**
$7^6 = 4$ **mod** 11	**6**	**7**
$7^7 = 6$ **mod** 11	**7**	**8**
$7^8 = 9$ **mod** 11	**8**	**9**
$7^9 = 8$ **mod** 11	**9**	**10**

It is seen that one value of x is mapped to one value of a.