Addition to last lecture:

I have just explained for the index-calculation how to calculate "log \( x \) \( (p_i) \)" of the factor base \( (p_i) \), but not how to calculate the discrete log \( \text{log}_x (\beta) \)

Therefore, on page 2 of the last lecture notes, before

1. Most efficiently it needs to be added:
   - Take random \( b \) until \( x^b \cdot \beta = \frac{e}{x} \cdot p_i \cdot \lambda_i \) can be found

2. \( b + \text{log}_x (\beta) = \sum \lambda_i \cdot \text{log}_x (p_i) - b \)
Remarks to probabilistic procedure of determining point on EC to a message \( M \).

- Obviously the procedure returns a point on an EC with high prob.

- If \( \frac{1}{2} \) half of the elements are quadratic residues, assuming that you may pick any quadratic residue with probability \( \frac{1}{2} \) (even if they are neighbors), the probability of failing to find one in the above alg is given by \( (\frac{1}{2})^k \).

If a point \((\epsilon, \gamma)\) on the EC is given, the corresponding original integer message will be \( M = \left\lfloor \frac{\epsilon}{2^k} \right\rfloor \).

Analogously to the deterministic approach, the message space needs to be reduced such that a unique demapping is possible.

13.4.3 ElGamal on elliptic curves (\( EC \) ElGamal)

\( a \) private key: random number \( x_a \in \{ 1, \ldots, n - 2 \} \)
\( a \) public key: \( x_a \cdot P \)

\( B \) wants to encrypt \( m \in \langle P \rangle \), therefore for getting on we may apply 13.4.2.

(i) \( B \) chooses random \( k \in \{ 2, \ldots, n - 2 \} \), computes \( Q = k \cdot P \)

(ii) \( B \) computes \( R = k (x_a \cdot P) + m \)

(iii) \( B \rightarrow A : (Q, R) \)

\( A \) deciphers:

(i) \( A \) computes \( x_a \cdot Q = x_a \cdot k \cdot P \)

(ii) \( A \) computes \( R - x_a \cdot Q = k (x_a \cdot P) + m - x_a \cdot k \cdot P = m \)

Use demapping of 13.4.2.
13.4.4 Elliptic Curve Integrated Encryption Scheme (ECIES)

ECIES is a variant of ElGamal introduced by Bellare, Rogaway. A Diffie-Hellman shared secret is used to derive two symmetric keys \( k_1 \) and \( k_2 \). \( k_1 \) is used for symmetric encryption and \( k_2 \) for ciphertext authentication.

We need the following primitives:
- Symmetric encryption function: \( \text{ENC}_{k_1} \) (e.g., AES)
  with corresponding decryption: \( \text{DECK}_{k_1} \)
- Message authentication code: \( \text{MAC}_{k_2} \) (e.g., HMAC)
- Key derivation function: \( \text{KDF} \)

\[ k_1, k_2 = \text{KDF}(S) = H(S, 0) \cup H(S, 1) \cup H(S, 2) \ldots \cup H(S, i) \]
until enough bits for \( k_1 \) and \( k_2 \) are generated. \( H \) is hash function.

\( p \), \( p \) prime, \( E : y^2 = x^3 + ax + b \) / \( \mathbb{F}_p \) \( P \in \mathbb{E}(\mathbb{F}_p) \)

\( \text{o} t. \ \text{and}(P) = n \) \( \text{prime} \) \( h = \# \mathbb{E}(\mathbb{F}_p) \)

\( \% \) key generation\% 
- Choose a random \( d \in \{2, \ldots, n-2 \} \) \( P \) \( \text{(private key)} \)
- Compute \( Q = dP \) \( \text{(public key)} \)

\( k \) encryption of \( m \in \{0, 1\}^* \)
1) Choose a random \( \kappa \in \{1, \ldots, n-2 \} \)
2) \( R \leftarrow \kappa \cdot P \) \( t \leftarrow h \cdot b \cdot \kappa \) \( Q \) \( \text{if } t = 0 \) \( \text{goto 1} \)
3) \( (k_1, k_2) = \text{KDF}(t, R) \) where \( t \) is the \( t \) coordinate of \( t \)
4) \( C = \text{ENC}_{k_1}(m) \) \( t = \text{MAC}_{k_2}(C) \)
5) Send \( (R \leftarrow C, \kappa) \)
Decryption

1) Assume the validity of $R := \begin{cases} R \neq 0 & R \in \mathbb{F}_p \\ \end{cases}$

2) $2 \in h.d\cdot R$, check whether $2 \neq 0$

3) $(k_1, k_2) = KP (t_2, R)$

4) $t' \in MAC_{k_2} (c)$, check whether $t' = t$

5) $m \in DECK_1 (c)$

To show that decryption works, only prove that $R$ is computed correctly:

$h.d \cdot R = h.d \cdot k \cdot p = h \cdot h \cdot d \cdot p = h \cdot h \cdot q$  

13.4.5 Elliptic Curve Digital Signature Algorithm (ECDSA)

BC : $E/\mathbb{F}_p$, generator $P$, ord$(P) = n$, with $n = \left\lceil \log_2 (\text{num}) \right\rceil$

and a hash function $h$.

$x$: private key $\{x \leftarrow \mathbb{Z} \cap \{1, \ldots, n-1\}$

$y = x \cdot P$ is the public key
Algorithm 14: Creating signature \((r, t)\) on a message \(m\)

**Input:** Message \(m\)

**Output:** A signature \((r, t)\) on \(m\)

\(e \in h(m)\)

\(t \in \text{the } L_n \text{ leftmost bits of } e\)

**Repeat**

**Repeat**

Select random \(1 \leq k \leq n - 1\) with \(\gcd(k, n) = 1\)

\((t_1, t_2) \in \mathcal{P}\)

\(r \equiv x_1 \mod n\)

Until \(r \neq 0\)

\(r \equiv b^{-r} (r + t - r) \mod n\)

Until \(\gcd(c, n) = 1\)

Return \((r, t)\)

Algorithm 15: Verifying signature \((r, t)\) on a message \(m\)

**Input:** Message \(m\) and \((r, t)\)

**Output:** Acceptance or denial of signature \((r, t)\) on message \(m\)

Verify that \(1 \leq r_1 \leq n - 1\)

\(e \in h(m)\)

\(t \in \text{the } L_n \text{ leftmost bits of } e\)

\(c \equiv r^{-1} \mod n\)

\(u_1 \equiv c \cdot e \mod n\)

\(u_2 \equiv r \cdot c \mod n\)

\((r_1, r_2) \equiv u_1 \cdot P + u_2 \cdot Y\)

if \(v \mod n = v\) then

return accept

else

return deny

end if
The verification process holds true as

\[
\langle \nu_1 \nu_2 \rangle = \mu_1 \cdot \nu + \mu_2 \cdot \gamma = \gamma^{-1} \cdot \nu + \gamma \cdot \nu \cdot x \cdot \nu
\]

\[
= \gamma^{-1} (\gamma + \gamma \cdot x) \cdot \nu = b \cdot \nu = (\nu - y) \equiv (\nu_1 \nu_2) \mod \nu