Security and Privacy

Crypto for e-voting protocols

28.05.2019

Ph. Oechslin

EPFL

Outline

- Multiplicative Groups
- ElGamal encryption
- Homomorphism
- Key sharing
- Zero knowledge proofs (ZKP)
- Mixnets

Multiplicative Groups

Multiplicative Groups

- What for: Group of numbers that represent votes, encrypted votes and can be used in proofs

- What: $\mathcal{G} = (G, \times, ^{-1}, 1)$
  - G: set of numbers
  - $\times$: multiplication operator
  - $^{-1}$: inverse of multiplication
  - 1: neutral element $\in G$
  - the multiplication of two elements of the group is always an element of the group (closure)

Multiplicative Groups

- Example: $\mathbb{Z}_p^*$ (integers modulo prime number p, * means without 0)
  - $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$
  - We don't want 0 because it has no inverse
  - We want p prime to avoid $a \times b = 0 \mod p$
  - $2 \times 5 \equiv 3 \pmod 7$ we just write $2 \times 5 = 3$
  - $2 \times 4 = 1,\ 2 \times 5 = 3,\ 3 \times 4 = 5$
  - $2^{-1} = 4$, $4^{-1} = 2$

- generator: g is a generator if $\{g^1, ..., g^{p-1}\} = G$
  - 3 is a generator of $\mathbb{Z}_7^*$: $\{3^1, 3^2, 3^3, 3^4, 3^5, 3^6\} = \{3, 2, 6, 4, 5, 1\}$

Multiplicative Groups

- Subgroups
  - $\mathbb{Z}_7^* = \{1,2,3,4,5,6\}$ has order 6 (6 elements)
  - $\mathbb{G}_3 = \{1,2,4\}$ is a subgroup of order 3 of $\mathbb{Z}_p^*$
  - $1 \times 2 = 2, \;\; 2 \times 2 = 4, \;\; 2 \times 4 = 1, \;\; 4 \times 4 = 2$
  - the order of a subgroup divides the order of the group
  - there is another subgroup, of order 2, can you see it?
  - if the group is of prime order, it has no subgroups !

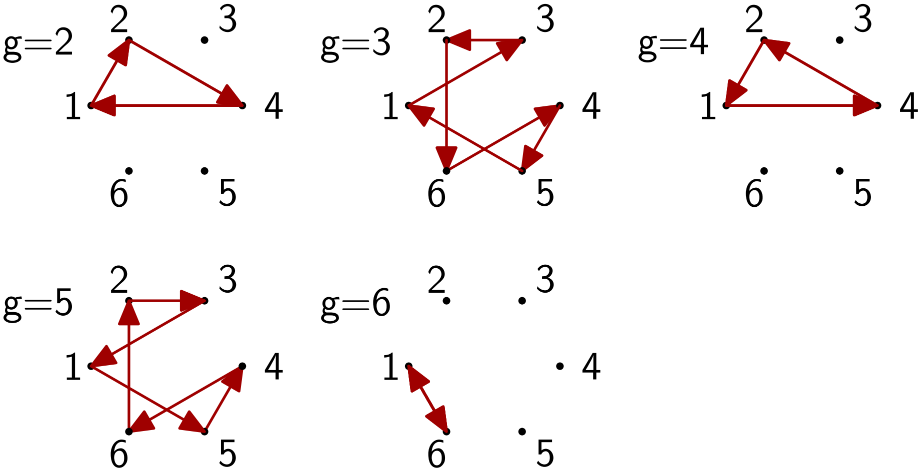
    ➡ any element of the group, except 1, is a generator
  - $\{1,2,4\}$ are the quadratic residues of $\mathbb{Z}_7^*$
  - $1^2 = 1,\; 2^2 = 4,\; 3^2 = 2,\; 4^2 = 2,\; 5^2 = 4,\; 6^2 = 1$

Multiplicative Groups

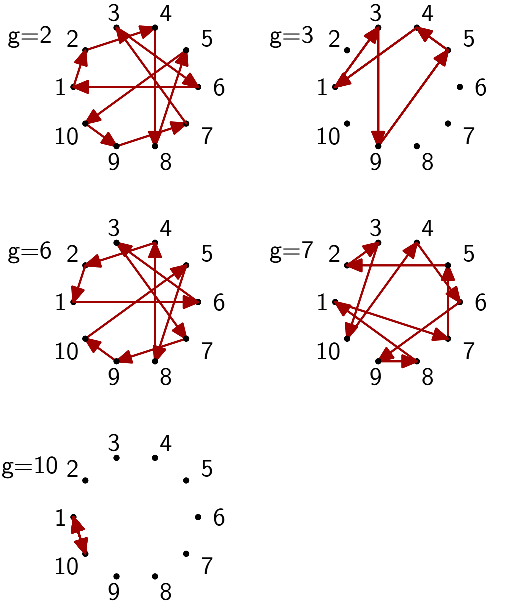
- For ElGamal encryption we need a prime order subgroup
- We start with p prime, but because we remove 0, the order of $\mathbb{Z}_p^*$ is pair. (since p is odd)
  - So $p-1$ at least has a factor 2. We want this to be the only one
- We choose $p = 2q+1$, where p and q are both primes.
  - (p is called a 'safe prime' and q a 'Sophie-Germain prime')
  - $p-1$ thus only has two factors: 2 and q
  - we have one subgroup or prime order q (and one of order 2)
- We work in a group $\mathbb{G}_q$ defined by p, q and the generator $g \in \mathbb{G}_q \backslash \{1\}$
- We can write any element of $\mathbb{G}_q$ as $g^{x \bmod q} \bmod p$
- We often omit to write $\bmod q$ and $\bmod p$

Some drawings

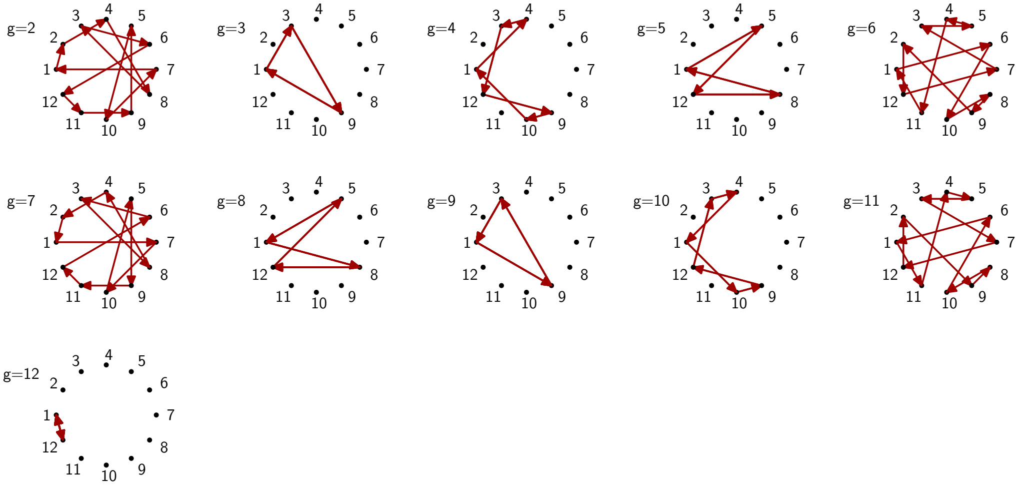
Generators and subgroups in $\mathbb{Z}_7^*$



Generators and subgroups in $\mathbb{Z}_{11}^*$

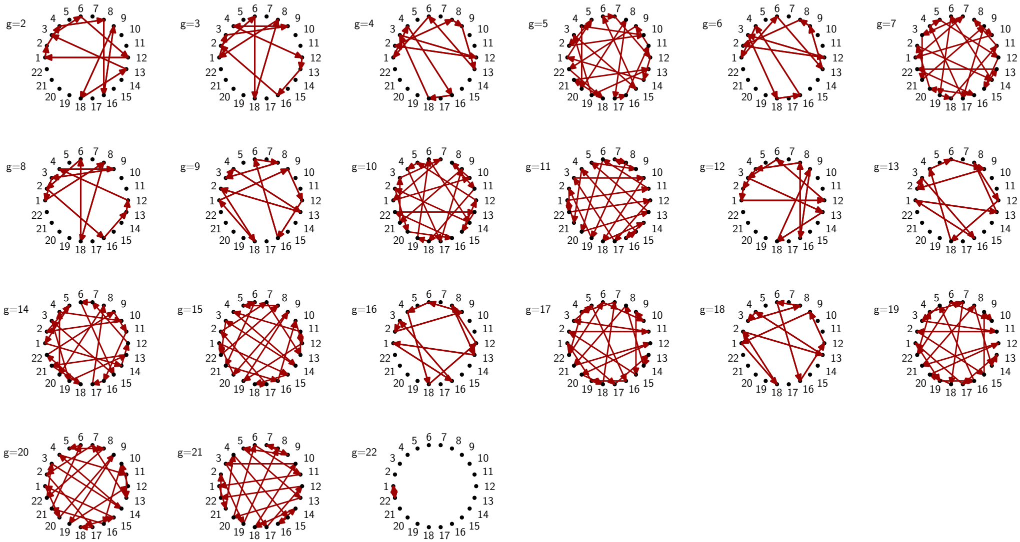


Generators and subgroups in $\mathbb{Z}_{13}^*$



- $\mathbb{Z}_{13}^*$ is of order 12, subgroup of order 6 has subgroups of 2,3
  - 13 is not a safe prime

Generators and subgroups in $\mathbb{Z}_{23}^*$



- $\mathbb{Z}_{23}^*$ is of order 22, subroup of order 11 is of prime order

ElGamal Encryption

ElGamal Encryption

- What for: Encrypt the votes in a way that we can use key shares for decrypting and homomorphism to make them anonymous

- How:
  - private key $sk \in \mathbb{Z}_q$
  - public key $pk = g^{sk} \in \mathbb{G}_q$
  - Encryption of message m with key pk and randomness r:
    - choose random r, multiply m with public key to the power of r
    - calculate g to the power of r

    $\texttt{Enc}_{pk}(m, r) = (m \cdot pk^r, g^r) = (a, b)$
  - Decryption:
    - divide a by b to the power of the secret key

    $\texttt{Dec}_{sk}(a, b) = a/b^{sk} = m$

✓ $\texttt{Dec}_{sk}(\texttt{Enc}_{pk}(m, r)) = m \cdot g^{sk^r}/g^{r^{sk}} = m$

ElGamal Encryption

- Homomorphism

- We can multiply encrypted values to get the encryption of the product:
  - $\mathtt{Enc}_{pk}(m_1, r_1) \cdot \mathtt{Enc}_{pk}(m_2, r_2)$

    $= (a_1 \cdot a_2, b_1 \cdot b_2) = (m_1 \cdot m_2 \cdot pk^{r_1+r_2}, g^{r_1+r_2})$

    $= \mathtt{Enc}_{pk}(m_1 \cdot m_2, r_1 + r_2)$

- Re-encryption: we can multiply by an encryption of one to keep the same value but change a and b:

$$\mathtt{Enc}_{pk}(m, r) \cdot \mathtt{Enc}_{pk}(1, r') = \mathtt{Enc}_{pk}(m, r + r')$$

Using key shares

Using key shares

- What for
  - reduce the trust need to have in single parts of the system
- How
- Use a private key made of shares
- Each share of the private key is held by a different entity
- To decrypt a message, all entities must collaborate
- As long as just one entity succeeds in protecting its share of the key, nothing bad can happen
  - We only need to trust that one out of n entities is honest.

El Gamal key shares

- s parties can create a key pair each $pk_j = g^{sk_j}$
- The public part can be multiplied to a single pulbic key: $pk = \prod_{j=1}^{s} pk_j$ which is equal to $g^{\sum_{j=1}^{s} sk_j}$
  - The private key is thus $\sum_{j=1}^{s} sk_j$
- pk can be used as is for encryption
- for decryption, each party calculates b^{sk_j}
  $\mathtt{Dec}_{sk}(a,b) = a/b^{\sum_{j=1}^{s} sk_j} = a \cdot /\prod_{j=1}^{s} b^{sk_j} = m$
- Note: the message can be decrypted without anybody knowing the complete private key!

El Gamal key shares

- For e-voting:
  - the voter uses the single public key to encrypt the vote,
  - the four CC calculate b^{sk_j}
  - the server multiplies the parts from the CC with a and recovers the vote.

Zero Knowledge Proofs

Zero Knowledge Proofs (ZKPs)

- What for: prove we know the content of an encrypted vote without revealing it, that we use the correct key for decryption, without revealing it, ...
- How:
  - Ask the prover to commit to a value.
  - Give him a challenge to solve using the committed value
  - solution is only possible if the prover knows a secret
  - solution does not reveal secret

Zero Knowledge Proofs

- Example
  - $y = g^x$
  - Prover P publishes y and pretends to know x
  - P commits to ω and publishes $t = g^\omega$
  - Verifier gives challenge c
  - P publishes $s = cx + \omega$
  - Verifier can verify that $g^s = ty^c$
    $g^s = g^{cx} g^\omega = g^\omega g^{xc}$
  - without knowing x, the prover could not calculate s
  - if he knew c before committing to t, he could chose random s and calculate corresponding t.

Non-Interactive ZKPs (NIZKPs)

- We don't want to send a challenge c to P. We let P generate it himself with a given hash function.
  - $y = g^x$
  - P publishes y and pretends to know x
  - P commits to ω and publishes $t = g^\omega$
  - P publishes challenge $c = H(y, t)$
  - P publishes $s = cx + \omega$
  - Verifier checks that $c = H(y, t)$
  - Verifier checks that $g^s = ty^c$

  - P has to commit to t before he can calculate the challenge
  - we write $\pi = NIZKP[(x) : y = g^x] = \{t, c, s\}$
    proof of knowledge of logarithm

NIZKP composition

- We can prove that a value is the same in two expressions g_1^x and g_2^x
- use two commitments of ω with g_1 and g_2
  - P commits to $(t_1, t_2) = ({g_1}^{\omega}, {g_2}^{\omega})$
  - P publishes a single challenge $c = H(y_1, t_1, y_2, t_2)$
  - P publishes $s = cx + \omega$
  - Verifier checks that $c = H(y_1, t_1, y_2, t_2)$
  - Verifier checks that $g_1^s = t_1 y^c$, $g_2^s = t_2 y^c$
  - we write : $\pi = NIZKP[(x) : y_1 = {g_1}^x \wedge y_2 = {g_2}^x] = \{t_1, t_2, c, s\}$
    proof of equality of logarithm

NIZKP+text = Schnorr Signature

- What for: we can add some text to a NIZKP. Typically, we can add the voting card ID into the proofs generated by the voter.
  - the voter signs his vote with his voting card number

- How: We just add the text into the hash function that creates the challenge from the commitment: $c = H(y, t, '\mathrm{sometext}')$
  - The NIZKP becomes a signature of the text.
  - $S = NIZKP[(x) : y = g^x, '\mathrm{text}'] = \{\mathrm{t}, \mathrm{c}, \mathrm{s}\}$

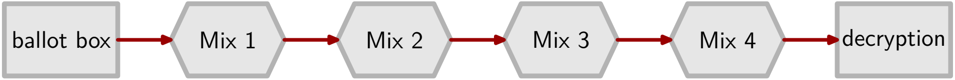
NIZKP: Summary

- Proof of knowledge of logarithm $NIZKP[(x) : y = g^x]$
  - When generating a keypair, a CC can prove it knows the private key
  - When doing ElGamal encryption, a voter can prove that he knows r in g^r or $m \cdot pk^r$. The voter can include his voting card ID into the proof
- Proof of equality of logarithm $NIZKP[(x) : y_1 = {g_1}^x \wedge y_2 = {g_2}^x]$
  - CCs can prove that the key used to decrypt a vote is the same as the private key corresponding to the public encryption key
  - A voter can prove that the encrypted ballot submitted to the ballot box contains the same votes as the ones for which verification codes are requested

Mixnets

Mixnets

- What for: We mix the ballots before they are decrypted in order to prevent that a decrypted vote can be traced back a vote submitted by a voter.
- How:
  - A mathematical operation is applied to the ballots to change their representation without changing their content
  - The ballots are reordered randomly (shuffled)
  - Each CC applies one mix operation. If at least one CC is honest, the mix can not be inversed.



Mixnets

- To make votes anonymous, each mixer multiplies the encrypted ballot with an encrypted neutral element.

$$\mathtt{Enc}_{pk}(m, r) \cdot \mathtt{Enc}_{pk}(1, r') = \mathtt{Enc}_{pk}(m, r + r')$$

- The mixnet must generate NIZKPs to prove
  - that the operation on the ballots did not change their content
  - that all ballots from the input are contained in the output

- It is challenge to create an efficient provable mixnet
  - chVote uses Wikström's mixnet & proofs
  - sVote uses Bayer and Groth

- The proofs are complicated in both cases...

Conclusions

Conclusions

- Homomorphic crypto systems are useful for e-voting
  - we can work on votes without decrypting them
  - e.g. we can anonymize them by multiplying by an encrypted 1
- With NIZKP we can prove
  - knowledge of a logarithm, equality of two logarithms
  - that a vote was cast with a given card id
  - that we correctly do the multiplication by one
  - that we correctly decrypt
- Mixnets combine both to provide vote secrecy while preserving correctness
- Splitting keys into shares allows reduce the required trust
  - we only need to trust 1 in n elements

References

- The specification of chvote (the protocol of GE) has a good introduction to this crypto **eprint.iacr.org/325.pdf**