Advanced cryptography and applications

Christian Grothoff

Berner Fachhochschule

4.6.2021

Learning Objectives

- Polkadot
- Homomorphic Encryption
- SMC: Scalar Product
- Blind Signatures

Guest Speaker: Dr. Jeffrey Burdges

- Recovering pure mathematician
- “Applied” cryptographer
- Lead protocol designers for Polkadot at Web3 foundation

Break

Homomorphic Encryption

$$E(x_1 \oplus x_2) = E(x_1) \otimes E(x_2) \quad (1)$$

Multiplicative Homomorphism: RSA & ElGamal

- Unpadded RSA (multiplicative):

$$E(x_1) \cdot E(x_2) = x_1^e x_2^e = E(x_1 \cdot x_2) \tag{2}$$

- ElGamal:

$$\begin{aligned} E(x_1) \cdot E(x_2) &= (g^{r_1}, x_1 \cdot h^{r_1})(g^{r_2}, x_2 \cdot h^{r_2}) && (3) \\ &= (g^{r_1+r_2}), (x_1 \cdot x_2) h^{r_1+r_2}) && (4) \\ &= E(m_1 \cdot m_2) && (5) \end{aligned}$$

Additive Homomorphism: Paillier

$$\mathsf{E}_K(m) := g^m \cdot r^n \mod n^2, \tag{6}$$

$$\mathsf{D}_K(c) := \frac{(c^\lambda \mod n^2) - 1}{n} \cdot \mu \mod n \tag{7}$$

where the public key $K = (n, g)$, m is the plaintext, c the ciphertext, n the product of $p, q \in \mathbb{P}$ of equal length, and $g \in \mathbb{Z}^*_{n^2}$. In Paillier, the private key is (λ, μ), which is computed from p and q as follows:

$$\lambda := \mathrm{lcm}(p-1, q-1), \tag{8}$$

$$\mu := \left(\frac{(g^\lambda \mod n^2) - 1}{n}\right)^{-1} \mod n. \tag{9}$$

Paillier offers additive homomorphic public-key encryption, that is:

$$\mathsf{E}_K(a) \otimes \mathsf{E}_K(b) \equiv \mathsf{E}_K(a+b) \tag{10}$$

for any public key K.

Fully homomorphic encryption

Additive:

$$E(A) \oplus E(B) = E(A + B) \quad (11)$$

and multiplicative:

$$E(A) \otimes E(B) = E(A \cdot B) \quad (12)$$

Known cryptosystems: Brakerski-Gentry-Vaikuntanathan (BGV), NTRU, Gentry-Sahai-Waters (GSW).

Break

Secure Multiparty Computation: Scalar Product

- Original idea by Ioannids et al. in 2002 [4] (use: $(a-b)^2 = a^2 - 2ab + b^2$)
- Refined by Amirbekyan et al. in 2007 (corrected math) [1]
- Now providing protocol with practical extensions (negative numbers, small numbers, set intersection).

Preliminaries

- Alice has public key A and input map $m_A : M_A \to \mathbb{Z}$.
- Bob has public key B and input map $m_B : M_B \to \mathbb{Z}$.
- We want to calculate

$$\sum_{i \in M_A \cap M_B} m_A(i) m_B(i) \tag{13}$$

- We first calculate $M = M_A \cap M_B$.
- Define $a_i := m_A(i)$ and $b_i := m_B(i)$ for $i \in M$.
- Let s denote a shared static offset.

Network Protocol

- Alice transmits $\mathsf{E}_A(s + a_i)$ for $i \in M$ to Bob.
- Bob creates two random permutations π and π' over the elements in M, and a random vector r_i for $i \in M$ and sends

$$
\begin{aligned}
R : &= \mathsf{E}_A(s + a_{\pi(i)}) \otimes \mathsf{E}_A(s - r_{\pi(i)} - b_{\pi(i)}) && (14)\\
&= \mathsf{E}_A(2 \cdot s + a_{\pi(i)} - r_{\pi(i)} - b_{\pi(i)}), && (15)\\
R' : &= \mathsf{E}_A(s + a_{\pi'(i)}) \otimes \mathsf{E}_A(s - r_{\pi'(i)}) && (16)\\
&= \mathsf{E}_A(2 \cdot s + a_{\pi'(i)} - r_{\pi'(i)}), && (17)\\
S : &= \sum (r_i + b_i)^2, && (18)\\
S' : &= \sum r_i^2 && (19)
\end{aligned}
$$

Decryption (1/3)

Alice decrypts R and R' and computes for $i \in M$:

$$a_{\pi(i)} - b_{\pi(i)} - r_{\pi(i)} = \mathrm{D}_A(R) - 2 \cdot s, \tag{20}$$

$$a_{\pi'(i)} - r_{\pi'(i)} = \mathrm{D}_A\left(R'\right) - 2 \cdot s, \tag{21}$$

which is used to calculate

$$T := \sum_{i \in M} a_i^2 \tag{22}$$

$$U := -\sum_{i \in M} (a_{\pi(i)} - b_{\pi(i)} - r_{\pi(i)})^2 \tag{23}$$

$$U' := -\sum_{i \in M} (a_{\pi'(i)} - r_{\pi'(i)})^2 \tag{24}$$

Decryption (2/3)

She then computes

$$\begin{aligned}
P &:= S + T + U \\
&= \sum_{i \in M} (b_i + r_i)^2 + \sum_{i \in M} a_i^2 + \left(- \sum_{i \in M} (a_i - b_i - r_i)^2 \right) \\
&= \sum_{i \in M} \left((b_i + r_i)^2 + a_i^2 - (a_i - b_i - r_i)^2 \right) \\
&= 2 \cdot \sum_{i \in M} a_i (b_i + r_i). \\
P' &:= S' + T + U' \\
&= \sum_{i \in M} r_i^2 + \sum_{i \in M} a_i^2 + \left(- \sum_{i \in M} (a_i - r_i)^2 \right) \\
&= \sum_{i \in M} \left(r_i^2 + a_i^2 - (a_i - r_i)^2 \right) = 2 \cdot \sum_{i \in M} a_i r_i.
\end{aligned}$$

Decryption (3/3)

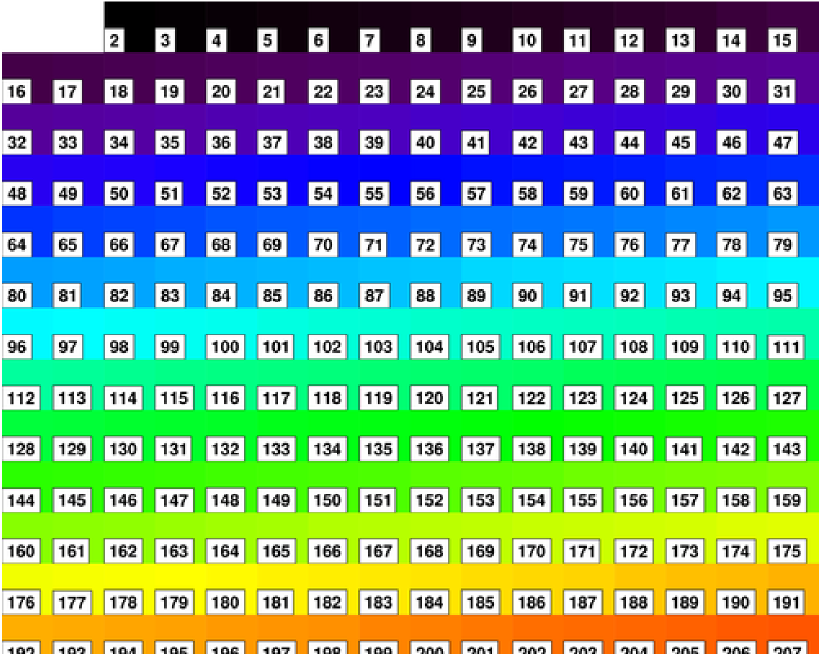
Finally, Alice computes the scalar product using:

$$\frac{P - P'}{2} = \sum_{i \in M} a_i(b_i + r_i) - \sum_{i \in M} a_i r_i = \sum_{i \in M} a_i b_i. \qquad (25)$$

Computing Discrete Logarithms

Who said calculating DLOG was hard?

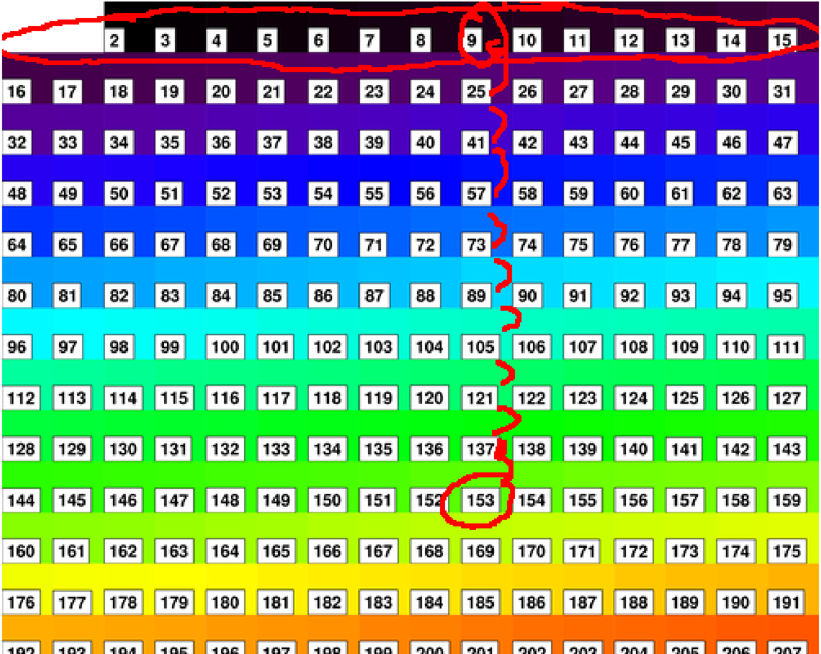
Computing Discrete Logarithms



Baby Steps


2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111
112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175
176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191

Giant Steps


2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111
112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127
128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175
176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191

ECC Version[1]

Alice's geheimer Wert sei a. Alice schickt an Bob $(g_i, h_i) = (g^{r_i}, g^{r_i a + a_i})$ mit zufälligen Werten r_i für $i \in M$.
Bob antwortet mit

$$\left(\prod_{i \in M} g_i^{b_i}, \prod_{i \in M} h_i^{b_i}\right) = \left(\prod_{i \in M} g_i^{b_i}, \left(\prod_{i \in M} g_i^{b_i}\right)^a g^{\sum_{i \in M} a_i b_i}\right)$$

Alice kann dann berechnen

$$\left(\prod_{i \in M} g_i^{b_i}\right)^{-a} \cdot \left(\prod_{i \in M} g_i^{b_i}\right)^a \cdot g^{\sum_{i \in M} a_i b_i} = g^{\sum_{i \in M} a_i b_i}.$$

Falls $\sum_{i \in M} a_i b_i$ ausreichend klein ist, kann Alice dann das Skalarprodukt durch Lösung des DLP bestimmen.

[1] Joint work with Tanja Lange

Performance Evaluation (AMD Threadripper 1950)

Length	RSA-2048	ECC-2^{20}	ECC-2^{28}
25	4 s	0.1 s	4.2 s
50	8 s	0.1 s	4.3 s
100	10 s	0.2 s	4.3 s
200	19 s	0.2 s	4.3 s
400	35 s	0.3 s	4.3 s
800	74 s	0.4 s	4.5 s
800	1234 kb	65 kb	65 kb

The pre-calculation of ECC-2^{28} is $\times 16$ more expensive than for ECC-2^{20} as the table is set to have size $\sqrt{n}$.

Exercise

Implement function to calculate DLOG.

Break

Reminder: RSA

Pick p, q prime and e such that

$$GCD((p-1)(q-1), e) = 1 \tag{26}$$

- Define $n = pq$,
- compute d such that $ed \equiv 1 \mod (p-1)(q-1)$.
- Let $s := m^d \mod n$.
- Then $m \equiv s^e \mod n$.

RSA Summary

- Public key: n, e
- Private key: $d \equiv e^{-1} \mod \phi(n)$ where $\phi(n) = (p-1) \cdot (q-1)$
- Encryption: $c \equiv m^e \mod n$
- Decryption: $m \equiv c^d \mod n$
- Signing: $s \equiv m^d \mod n$
- Verifying: $m \equiv s^e \mod n$?

Low Encryption Exponent Attack

- e is known
- M maybe small
- $C = M^e < n$?
- If so, can compute $M = \sqrt[e]{C}$

$\Rightarrow$ Small e can be bad!

Padding and RSA Symmetry

- Padding can be used to avoid low exponent issues (and issues with $m = 0$ or $m = 1$)
- Randomized padding defeats chosen plaintext attacks
- Padding breaks RSA symmetry:

$$D_{A_{priv}}(D_{B_{priv}}(E_{A_{pub}}(E_{B_{pub}}(M)))) \neq M \tag{27}$$

- PKCS#1 / RFC 3447 define a padding standard

Blind signatures with RSA [3]

1. Obtain public key (e, n)
2. Compute $f := FDH(m)$, $f < n$.
3. Pick blinding factor $b \in \mathbb{Z}_n$
4. Transmit $f' := fb^e \mod n$

Blind signatures with RSA [3]

1. Obtain public key (e, n)
2. Compute $f := FDH(m)$, $f < n$.
3. Pick blinding factor $b \in \mathbb{Z}_n$
4. Transmit $f' := fb^e \mod n$

1. Receive f'.
2. Compute $s' := f'^d \mod n$.
3. Send s'.

Blind signatures with RSA [3]

1. Obtain public key (e, n)
2. Compute $f := FDH(m)$, $f < n$.
3. Pick blinding factor $b \in \mathbb{Z}_n$
4. Transmit $f' := fb^e \mod n$

1. Receive f'.
2. Compute $s' := f'^d \mod n$.
3. Send s'.

1. Receive s'.
2. Compute $s := s'b^{-1} \mod n$

References

Artak Amirbekyan and Vladimir Estivill-castro.
A new efficient privacypreserving scalar product protocol.
In *in Proc. of AusDM '07*, pages 209–214.

Mikhail J. Atallah and Wenliang Du.
Secure multi-party computational geometry.
Technical Report 2001-48, Purdue University, West Lafayette, IN 47907, 2001.

David Chaum.
Blind Signature System, pages 153–153.
Springer US, Boston, MA, 1984.

Ioannis Ioannidis, Ananth Grama, and Mikhail J. Atallah.
A secure protocol for computing dot-products in clustered and distributed environments.
In *31st International Conference on Parallel Processing (ICPP 2002), 20-23 August 2002, Vancouver, BC, Canada*, pages 379–384. IEEE Computer Society, 2002.