TTM4135 Worksheet 6: DLOG and Digital Signatures

Tjerand Silde and Emil August Hovd Olaisen
{tjerand.silde, emil.august.olaisen}@ntnu.no

Spring 2026

1. Review the definitions of the following concepts:
   (a) ElGamal cryptosystem
   (b) digital signature
   (c) existential forgery and selective forgery
   (d) Schnorr, DSA, ECDSA

2. Let $p = 43$. Verify that $g = 3$ is a generator of $\mathbb{Z}_p^*$. Suppose that Alice and Bob execute the Diffie–Hellman key exchange protocol with Alice's input being $a = 5$ and Bob's input $b = 13$. Show that both Alice and Bob will compute the same shared secret.

   Solution:
   We first factorize $p-1$ as $42 = 2 \cdot 3 \cdot 7$. To check that g is a generator we need to verify that $g^{21} \bmod 43 \neq 1$, $g^{14} \bmod 43 \neq 1$ and $g^6 \bmod 43 \neq 1$. Since $3^{21} \bmod 43 = 42$, $3^{14} \bmod 43 = 36$ and $3^6 \bmod 43 = 41$ it follows that 3 is indeed a generator.

   - Alice sends $g^a \bmod p = 3^5 \bmod 43 = 28$.
   - Bob sends $g^b \bmod p = 3^{13} \bmod 43 = 12$.
   - Bob computes $(g^a)^b \bmod p = 28^{13} \bmod 43 = 34$.
   - Alice computes $(g^b)^a \bmod p = 12^5 \bmod 43 = 34$.

3. Suppose that Alice has a public key $y = 5, g = 2, p = 11$ for the ElGamal encryption algorithm. Here $g = 2$ is a generator for $\mathbb{Z}_{11}^*$. Compute a valid ciphertext for the message $M = 3$ intended for Alice, showing the steps required.

   Solution: Choose a random k. Here we assume $k = 3$. Then compute $g^k \pmod p = 2^3 \pmod{11} = 8$ and $(c_1, c_2) = (g^k \bmod p, My^k \pmod p) = (8, 3 \cdot 5^3 \pmod{11}) = (8, 3 \cdot 4) = (8, 1)$. To check this we can decrypt.
   The private key is $x = 4$ (we can find by trial and error here). So the decrypted ciphertext is $c_2 \cdot (c_1^x)^{-1} = 1 \cdot (8^4)^{-1} \pmod{11} = 4^{-1} \pmod{11} = 3$.

4. Compare the efficiency of the ElGamal cryptosystem in $\mathbb{Z}_p^*$ and the RSA cryptosystem with modulus n. Assume that the size of the modulus p and n is the same in each case. Compare:

   - the cost of key generation;

- the computation required for encryption;
- the computation required for decryption;
- the size of the public keys and ciphertexts.

Solution:

In this answer we assume that the ElGamal cryptosystem is implemented in $\mathbb{Z}_p^*$. When implemented on elliptic curves it is much more efficient.

- Key generation for ElGamal requires obtaining a prime p and a generator g for $\mathbb{Z}_p^*$. Prime generation requires testing for primality which can be done efficiently using a probabilistic test such as the Miller–Rabin test. Checking for a generator requires a few exponentiations (depending on the number of factors of $p-1$). After p and g are found one more exponentiation is required to obtain the public key from the private key.

  Key generation for RSA requires finding two primes p and q, but these are only half the size of the modulus so finding them is cheaper than in ElGamal. After p and q are found the exponent d must be constructed from e which requires finding an inverse modulo $\phi(n)$. This can be done cheaply using the Euclidean algorithm.

  Overall key generation for RSA should be cheaper than for ElGamal, although they are not greatly different and key generation only happens once. If ElGamal reuses the p and g values then key generation is much cheaper. As we know, the modulus for RSA cannot be re-used.

- Encryption for ElGamal requires two exponentiations and one multiplication. Encryption for RSA requires one exponentiation. Moreover the exponent e can be chosen to be small – if $e = 3$ then encryption needs only two multiplications. Therefore RSA encryption is much cheaper than ElGamal encryption.

- Decryption for ElGamal requires one exponentiation, one multiplication and one inversion. An inversion can be done efficient using the Euclidean algorithm (or an exponentiation). RSA decryption requires one exponentiation. This exponentiation can be computed more cheaply by using the Chinese remainder theorem.

- ElGamal ciphertexts are twice the length of the modulus, while RSA ciphertexts are equal to the length of the modulus. Note that ElGamal plaintexts can be any value less than the modulus, but RSA requires padding which uses several bytes and means that messages have to be significantly shorter than the modulus.

5. In 2019 elections for the Moscow city parliament an electronic voting system used a simple variant of ElGamal encryption to protect a user vote M. For the public key $K = (K_1, K_2, K_3) = (y_1, y_2, y_3) = (g^{x_1}, g^{x_2}, g^{x_3})$ the following details the encryption algorithm. First the encrypting party chooses random k_1, k_2, k_3 and computes the following values.

   $c_1 = E(M, K_1) = (g^{k_1} \bmod p, M \cdot y_1^{k_1} \bmod p) = (a_1, b_1)$

   $c_2 = E(a_1, K_2) = (g^{k_2} \bmod p, a_1 y_2^{k_2} \bmod p) = (a_2, b_2)$

   $c_3 = E(a_2, K_3) = (g^{k_3} \bmod p, a_2 y_3^{k_3} \bmod p) = (a_3, b_3)$

   Finally the ciphertext is the 4-tuple $c = (b_1, b_2, a_3, b_3)$. Note that c_1, c_2 and c_3 are ordinary ElGamal ciphertexts.

   For unclear reasons, the implementation used a prime p with only 256-bits for computations in $\mathbb{Z}_p^*$. You can read the details of the analysis here: https://arxiv.org/abs/1908.05127.

   (a) Explain how the message M can be recovered from c given the private key (x_1, x_2, x_3)

   (b) How long is the private key?

(c) If an attacker can find discrete logarithms in $\mathbb{Z}_p^*$ in time T, how long will take the same attacker to find the message M?

Solution:

(a) Since c_1, c_2 and c_3 are ordinary ElGamal ciphertexts they can be decrypted sequentially with the separate parts of the private key. The decryptor first obtains a_2 from C_3 so that C_2 is known, then obtains a_1 from c_2 so that c_1 known and finally M that c_1.

(b) $3 \times 256 = 768$ bits.

(c) $3T$. You would compute k_3 from a_3, and then compute $a_2 = b_3(y_3^{k_3})^{-1}$. Next, you would compute k_2 from a_2, and $a_1 = b_2(y_2^{k_2})^{-1}$. Lastly, you would obtain k_1 from a_1 and compute $M = b_1(y_1^{k_1})^{-1}$.

6. Suppose an attacker can break the hash function H used to form a digital signature (RSA or DSA) by finding collisions. How can this lead to attacks on the signature? Is this attack existential or selective?

Solution: Signatures of two messages with the same hash value are the same. If an attacker finds a collision between messages M_1 and M_2 he may be able to persuade the signer to sign M_1, especially if it is a meaningful message. Thus he also obtains a forged signature on M_2. If we require security against a chosen message attack, then we assume the adversary can obtain signatures on any message of his choice so this attack is valid. This is generally an existential forger only.

This same process shows that breaking the second-preimage resistance of the hash function leads to a selective forgery.

7. Suppose that RSA signatures are used with a hash function that is not one-way (that is, the attacker can invert the hash function). Show how an existential forgery is possible against such a signature: an attacker can form valid signatures from e and n alone.

Solution: Recall that a valid RSA signature on a message M is the value $S = \mathsf{H}(M)^d \bmod n$ where d is the private signing key and n is the modulus. Then e is the public verification key.

The attacker chooses a random value X and computes $Y \equiv X^e \bmod n$. By the RSA equation, $Y^d \equiv X^{ed} \equiv X \pmod n$. Now suppose that the attacker can invert the hash function to find a value M with $\mathsf{H}(M) = Y$. Then we have $X = \mathsf{H}(M)^d \bmod n$ which means that X is the RSA signature for M.

8. (a) Show that the verification equation for DSA signatures works.
   (b) Similarly check that the verification equation works for ECDSA signatures.

Solution:

(a) We have $r = (g^k \bmod p) \bmod q$ and $s = k^{-1}(\mathsf{H}(M) - xr) \bmod q$ and $y = g^x \bmod p$. During the verification we compute $u_1 = \mathsf{H}(M)w \bmod q$ and $u_2 = rw \bmod q$. Therefore

$$\begin{aligned}
(g^{u_1} y^{-u_2} \bmod p) \bmod q &= (g^{\mathsf{H}(M)w \bmod q}(g^x)^{-rw \bmod q} \bmod p) \bmod q \\
&= (g^{w(\mathsf{H}(M)-xr) \bmod q} \bmod p) \bmod q \\
&= (g^{wks \bmod q} \bmod p) \bmod q \\
&= (g^k \bmod p) \bmod q \text{ since } w = s^{-1} \bmod q \\
&= r
\end{aligned}$$

(b) We have $(r, y) = [k]G$ for some y and $s = k^{-1}(e + rd) \bmod n$ and $Y = [d]G$, where $e = \mathsf{H}(M)$. During the verification we compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$. Therefore

$$\begin{aligned} [u_1]G + [u_2]Y &= [ew]G + [rwd]G \\ &= [w(e + rd)]G \\ &= [wks]G \\ &= [k]G \text{ since } w = s^{-1} \bmod n \\ &= (r, y) \end{aligned}$$

Thus the x-coordinates of $[u_1]G + [u_2]Y$ and $[k]G$ are the same.

9. Suppose the parameters $p = 23$, $q = 11$, $g = 3$ are used for the DSA signature.
   (a) Show that g has order q as required.
   (b) If the private key is $x = 5$, what is the public key y?
   (c) Compute a valid signature for a message m whose hash value is assumed to be $\mathsf{H}(M) = 8$.
   (d) Show that the verification equation works for your signature.

Solution:

(a) To show that 3 has order 11 we need to check that $3^{11} \equiv 1 \pmod{23}$ but it is not true that $3^j \equiv 1 \pmod{23}$ for $1 < j < 11$.

j	$3^j \pmod{23}$
1	3
2	9
3	4
4	12
5	13
6	16
7	2
8	6
9	18
10	8
11	1

(b) From the above table we see that the public key is $y = g^x \pmod{p} = 3^5 \pmod{23} = 13$.

(c) To sign the message with $\mathsf{H}(M) = 8$ we need to choose a random value k. The simplest case to do by hand is $k = 2$ (but this would of course not be secure in practice). Then the signature has two parts r and s.

$$\begin{aligned} r &= (g^k \bmod p) \bmod q \\ &= (3^2 \bmod 23) \bmod 11 \\ &= 9 \bmod 11 \end{aligned}$$

To obtain s first note that $k^{-1} \pmod{11} \equiv 6$. Then

$$\begin{aligned} s &= k^{-1}(\mathsf{H}(M) - xr) \bmod q \\ &= 6(8 - 5 \cdot 9) \bmod 11 \\ &= 6 \cdot 7 \bmod 11 \\ &= 9 \end{aligned}$$

Therefore a valid signature is $(r, s) = (9, 9)$.

(d) To verify the signature we first calculate $w \equiv s^{-1} \bmod q \equiv 9^{-1} \bmod 11 = 5$. Next we compute $u_1 \equiv \mathsf{H}(M) \cdot w \bmod q = 8 \cdot 5 \bmod 11 = 7$ and $u_2 \equiv r \cdot w \bmod q \equiv 9 \cdot 5 \bmod 11 = 1$. Finally we need to check that

$$\begin{aligned}
(g^{u_1} y^{-u_2} \bmod p) \bmod q &= r \bmod q \\
(g^{u_1} y^{-u_2} \bmod p) \bmod q &\equiv (3^7 \cdot 13^{-1} \bmod 23) \bmod 11 \\
&\equiv (3^7 \cdot 16 \bmod 23) \bmod 11 \\
&\equiv (2 \cdot 16 \bmod 23) \bmod 11 \\
&\equiv (32 \bmod 23) \bmod 11 \\
&\equiv 9 \bmod 11 \\
&\equiv r
\end{aligned}$$

Thus $(r, s) = (9, 9)$ is a valid signature.

10. Compare the efficiency of DSA signatures, ElGamal signatures, and RSA signatures assuming that the modulus size is 2048 bits in each case, and that the prime q in DSA signatures is of length 256 bits. Compare:

- the cost of key generation;
- the computation required for signature generation;
- the computation required for signature verification;
- the size of the public keys and signatures.

Solution:

- Key generation for RSA signatures and ElGamal signatures is exactly the same as for RSA encryption and ElGamal encryption respectively. In general RSA key generation is cheaper, but if we assume that parameters p and g are fixed for ElGamal signatures, then ElGamal key generation is cheaper.

  The difference between key generation for ElGamal and DSA signatures is that the private key for DSA signatures is chosen in the range $0 < x < q$ whereas for ElGamal signatures $0 < x < p - 1$. The generation of the public key $y = g^x \pmod p$ is therefore cheaper for DSA signatures than for ElGamal signatures. (Remember q is 256 bits long while p is 2048 bits long).

- – Signature generation for RSA signatures requires one full length exponentiation.
  – Signature generation for ElGamal signatures requires one full length exponentiation plus an inversion and a couple of multiplications.
  – Signature generation for DSA requires one short exponentiation plus one inversion (with a short modulus q) and a couple of multiplications.

  Overall DSA is easily the cheapest. Consider the example figures and assume that the square-and-multiply algorithm is used for exponentiation, and that squaring and multiplication are around the same cost. Then the RSA signature generation cost is around $2048 + 1024 = 3072$ multiplications. Even if we use an inefficient method for computing inverses (compute $k^{-1} \bmod q = k^{q-1} \bmod q$) the cost of DSA signature generation is only $384 + 384 + 2 = 770$ multiplications on average.

- – Signature verification for RSA signatures requires one full length exponentiation. However, in practice the public exponent e is chosen to be very short so that a very short exponentiation is all that is needed.

- Signature verification for ElGamal signatures requires three full length exponentiations. In practice this can be reduced to less than 2, but it is still more expensive than RSA in every case.
- Signature verification for DSA requires two short exponentiations plus one inversion (with a short modulus) and a couple of multiplications.

Overall when using a small modulus (particularly $e = 3$) RSA is the cheapest. If RSA had to use a random public exponent e then DSA would be the cheapest.

- The public key for RSA signatures is the pair (n, e). If e is short (and fixed for the whole system) then only n is required. In the best case this is 2048 bits.

  The public key for both ElGamal and RSA signatures is of the form $y = g^x \pmod p$ which is 2048 bits in our example. (Note that a shorter private key x in DSA does not result in a shorter public key y.)

  - RSA signatures are the same length as the modulus n, so 2048 bits in our example.
  - ElGamal signatures are two values of the same length as p, so 4096 bits in our example.
  - DSA signatures are two values of length q, so 512 bits in our example.

  Overall DSA signatures are by far the smallest option.

11. Suppose that the same value of the random k is used to generate two different DSA signatures. Show that this is sufficient for an attacker to find the private signing key. (This was the attack used to break the software verification on the Sony PlayStation 3 in 2010 because their implementation used a fixed k. Sony used the elliptic curve version: https://arstechnica.com/gaming/2010/12/ps3-hacked-through-poor-implementation-of-cryptography.)

Solution:

If (r_1, s_1) and (r_2, s_2) are two DSA signatures on the messages M_1 and M_2 with the same k, then:

$$\begin{aligned} s_1 &= k^{-1}(\mathsf{H}(M_1) - xr_1) \bmod q \\ s_2 &= k^{-1}(\mathsf{H}(M_2) - xr_2) \bmod q \end{aligned}$$

where x is the private signing key. So we have two equations with two unknowns, x and k. Note that we also have $r_1 = r_2 = r$. Since s_1 and s_2 are known, the attacker can compute

$$X = \frac{s_1}{s_2} = \frac{\mathsf{H}(M_1) - xr_1}{\mathsf{H}(M_2) - xr_2} \bmod q.$$

Thus

$$X(\mathsf{H}(M_2) - xr_2) = \mathsf{H}(M_1) - xr_1 \bmod q$$

or

$$x(r_1 - Xr_2) = \mathsf{H}(M_1) - X\mathsf{H}(M_2) \bmod q$$

and finally

$$x = \frac{\mathsf{H}(M_1) - X\mathsf{H}(M_2)}{(r_1 - r_2X)} \bmod q$$

where everything on the right is known and so we can solve for the private key x.

12. Recall that ECDSA signatures consist of a pair of value (r, s)

    (a) Show that if (r, s) is valid ECDSA signature for a message M then $(r, -s)$ is also a valid signature for M.

(b) Show that the ECDSA signature verification equation will be satisfied by the values $(r, s) = (0, 0)$ for *any* message M, as long as other checks are omitted. You can read about why this property led to a huge vulnerability in Java cryptography known as *psychic signatures*.

Solution:

(a) This question relies on knowing that for an elliptic curve point $R = (x, y)$, the inverse point is $-R = (x, -y)$. Consider the verification process shown in the lecture, and denote the values computed for $(r, -s)$ as dashed versions of the values computed for (r, s). Then we see that $u_1' = -u_1$ and $u_2' = -u_2$. So the elliptic curve point computed is $(x', y') = -(x, y) = (x, -y)$. Thus $x' = x = r$ and the verification is satisfied.

(b) Looking at the verification process in the slides, we can see that if $(r, s) = (0, 0)$ then $u_1 = 0$ and $u_2 = 0$ (this is assuming $0^{-1} \bmod n$ will be computed as 0, such as happened in the broken Java implementation). Then the (x, y) point on the curve will be computed as $(0, 0)$ and so $x = 0 = r$.