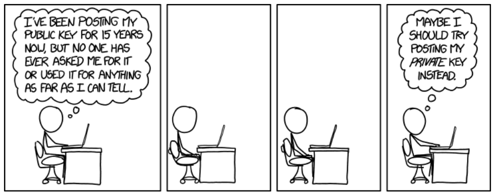
CPSC 418/MATH 318 Introduction to Cryptography

More on Message Authentication Codes, Public Key Cryptography, RSA

Renate Scheidler

Department of Mathematics & Statistics
Department of Computer Science
University of Calgary

Week 8



Outline

Message Authentication Codes – Recap

Definition 1 (Message authentication code (MAC))

A single-parameter family $\{MAC_K\}_{K\in\mathcal{K}}$ of many-to-one functions $MAC_K : \mathcal{M} \to \{0,1\}^n$ ($n \in \mathbb{N}$) satisfying:

- *Ease of computation with knowledge of K:* For any $M \in \mathcal{M}$ and $K \in \mathcal{K}$, $MAC_K(M)$ is easy to compute.
- *Computation resistance*: for any $K \in \mathcal{K}$, given zero or more message/MAC pairs $(M_i, MAC_K(M_i))$, it is computationally infeasible to compute any new message/MAC pair $(M, MAC_K(M))$, $M \neq M_i$ for all i, without knowledge of K.

Computation resistance provides message integrity (infeasible to generate a valide message/MAC pair $(M, MAC_K(M))$ without knowledge of K.

MAC Constructions

From a block cipher:

- Apply block cipher to message using a suitable mode of operation
- Last encrypted block is the MAC tag
- CBC-MAC, CFB-MAC, CMAC

Can also construct MACs from hash functions by incorporating a key.

MACs from Hash Functions

Basic idea: $MAC = H(M, K)$ where H is a cryptographically secure hash function and K is a secret key.

Advantage over CMAC: hash functions are faster than block ciphers.

Approaches:

- $MAC = H(M \| K)$: insecure if H is iterated
- $MAC = H(K \| M)$: similar problem
- $MAC = H(K_1 \| M \| K_2)$: "Sandwich MAC" — better, but potentially also vulnerable
- $MAC = H(K_1 \| H(K_2 \| M))$: Bellare, Canetti, Krawczyk (CRYPTO 1996) — HMAC

HMAC

Hash based message authentication code (HMAC):

$$HMAC_K(M) = H((\underbrace{K^+ \oplus \texttt{opad}}_{K_1}) \parallel H(\underbrace{K^+ \oplus \texttt{ipad}}_{K_2} \| M))$$

Description (assume H has compression function f and operates on b-bit blocks, eg. $b = 512$ for SHA-1):

1. $K^+ = 0 \ldots 0K$ (0-bits prepended to K so K^+ has b bits)
2. $K_1 = K^+ \oplus \text{opad}$, with $\text{opad} = (01011100)_{b/8}$
3. $K_2 = K^+ \oplus \texttt{ipad}$, with $\texttt{ipad} = (00110110)_{b/8}$
4. $T = H(K_2 \| M)$ (note that $f(IV, K_2)$ can be precomputed)
5. $HMAC_K(M) = H(K_1 \| T)$ (note that $f(IV, K_1)$ can be precomputed)

See FIPS 198.

Properties of HMAC

- $K_1 = K^+ \oplus \texttt{ipad}$ and $K_2 = K^+ \oplus \text{opad}$ are two pseudorandom keys derived from K by flipping some bits in the padded key K^+.
- Only three additional executions of f needed compared with just hashing M
  - Only one if key-dependent precomputations are used
- Provably secure; security is equivalent to one of the following:
  - Computing an output of f assuming IV is unknown
  - Finding collisions in H assuming IV is unknown.
- A birthday attack based on the second scenario is possible:
  - Significantly more difficult than ordinary birthday attack
  - Requires a MAC-generating oracle to compute valid message/MAC pairs because IV is secret

Details and a diagram in the HMAC handout on our "handouts" page.

MAC from SHA-3

KMAC (*Keccak Message Authentication Code*) is a SHA-3 derived message authentication method:

- Concatenates K with M and desired hash length m and passes concatenation to the SHA-3 derived hash function cSHAKE:

$$KMAC_K(M) = cSHAKE(K \| M \| m).$$

- Not vulnerable to the attacks on prepend construction $MAC_K = H(K \| M)$ for iterated hash functions H.
- Can also be used as a *pseudorandom function.*

See NIST SP 800-185.

Authenticated Encryption

MACs can be combined with block ciphers to provide confidentiality and message integrity via *authenticated encryption*.

"Encrypt-then-MAC": send $C \| MAC_{K'}(C)$ where $C = E_K(M)$

- Formally secure (Bellare-Namprempre 2007) since it preserves the integrity of the ciphertext and protects against *malleability*
- Prone to implementation errors (*e.g.* problem in IPSec found by Ferguson-Schneier 2003)

"MAC-then-encrypt": send $E_K(M \| MAC_{K'}(M))$

- More natural, less error-prone
- Can be more practical — if encryption is defeated or obviated, message integrity remains preserved
- There is also a "hash-then-encrypt" alternative

Why MACs?

Why use MACs, as opposed to "hash-then-encrypt" $C = E_K(M \| H(M))$?

- Sometimes only integrity is needed (no secrecy).

Why use "MAC-then-encrypt" versus "encrypt-then-MAC"?

- May need integrity longer than encryption (eg. archival use).
- Used in older versions of SSL/TLS.

Another variant is "MAC-and-encrypt" — send $C = E_K(M) \| MAC_{K'}(M)$

- Integrity and security are completely decoupled.
- Used in older versions of SSH.

Data Integrity via Modes of Operation

Combining cryptographic primitives (*e.g.* encryption with MACs) can be wrought with problems.

Block cipher modes of operation with built-in authentication are a more recent alternative approach.

- E.g. GMAC and *Galois Counter Mode* (GCM), derived from CTR mode; see NIST SP 800-38D
- GCM uses arithmetic modulo $x^{128} + x^7 + x^2 + x + 1$ (same polynomial as in CMAC)
- Used in TLS 1.3 (see IETF RFC 8446, `https://tools.ietf.org/html/rfc8446`, 2018)
- Another authentication mode is CCM (counter with CBC-MAC; see NIST SP 800-38C)

Attacks on MACs

The objective of adversary is to defeat computation resistance:

- Given zero or more pairs $(M_i, MAC_K(M_i))$, $i = 1, 2, \ldots$, compute a new message/MAC pair $(M, MAC_K(M))$ for some message $M \neq M_i$ for all i, without knowledge of K.
- Known-message, chosen-message, and adaptive-chosen-message variations are possible.

A more ambitious adversarial goal is to find the MAC key K.

Exhaustive Search Attack on MAC Space

Assume n-bit MACs, m-bit keys.

Attack:

- Pick a message, guess the MAC value (probability 2^{-n} of being correct).
- Requires "black-box" MAC verifier to confirm guesses.
- Expected number of attempts is 2^n.
- Does not find the MAC key.

Exhaustice Search Attack on Key Space

Assumes $m > n$ (longer keys than MACs, reasonable). This is a KPA:

- Given M_1 and $MAC_1 = MAC_K(M_1)$, compute $MAC_{i1} = MAC_{K_i}(M_1)$ for all possible keys K_i $(1 \leq i \leq 2^m)$.
- Expect 2^{m-n} keys to produce a match $MAC_1 = MAC_{i1}$ (2^m MACs produced, only 2^n possible MACs).
- Repeat on all matches with M_2 and $MAC_2 = MAC_K(M_2)$, reducing the number of possible keys to 2^{m-2n}. Iterate with M_j and $MAC_j = MAC_K(M_j)$, $j = 3, 4, \ldots$

Requirements:

- $\lceil m/n \rceil$ message/MAC pairs (m/n rounded up)
- $\lceil m/n \rceil \cdot 2^m$ MAC computations, but these can be conducted off-line if $M_1, M_2, \ldots$ are known in advance.

Summary on MAC Attacks

Brute-force attacks (n-bit MACS, m-bit keys):

1. 2^n to defeat computation resistance (find a valid message/MAC pair)
2. $\lceil \frac{m}{n} \rceil 2^m$ to find a MAC key

As usual, this should be best possible.

Cryptanalytic attacks also possible:

- For CMAC, one can try to attack the underlying block cipher.
- For HMAC and KMAC, one can try to attack the underlying hash function.

Were are we at?

Recall cryptographic services:

- Data confidentiality: discussed, also next
- Data integrity: discussed
- Authentication: discussed for data
- Non-repudiation
- Access Control: discussed a bit

Recall cryptographic mechanisms:

- Encryption — for confidentiality and limited data integrity: discussed, also next
- Hash functions, Message Authentication Codes (MACs) — for data integrity: discussed
- Digital signatures — for data origin authentication and non-repudiation
- Authentication protocol — for entity authentication

Back to Cryptographic Key Agreement

Recall efficient solutions to the key establishment problem:

1. Diffie-Hellman key agreement protocol
2. Public key cryptography — next!
   - Also used for authentication — later!

Public-Key Cryptography

Whitfield Diffie and Martin Hellman, "New Directions in Cryptography", 1976.

- Note that Diffie and Hellman did not describe a specific means of *implementing* a public-key cryptosystem.
- They merely described how one could be used to achieve security, authentication, (and indirectly, integrity and non-repudiation).

Public key crypto was also secretly discovered in 1970 as "non-secret encryption" by James H. Ellis of the UK's Government Communications Headquarters (GCHQ)

- Disclosed in 1987; see `http://cryptocellar.org/cesg/possnse.pdf`

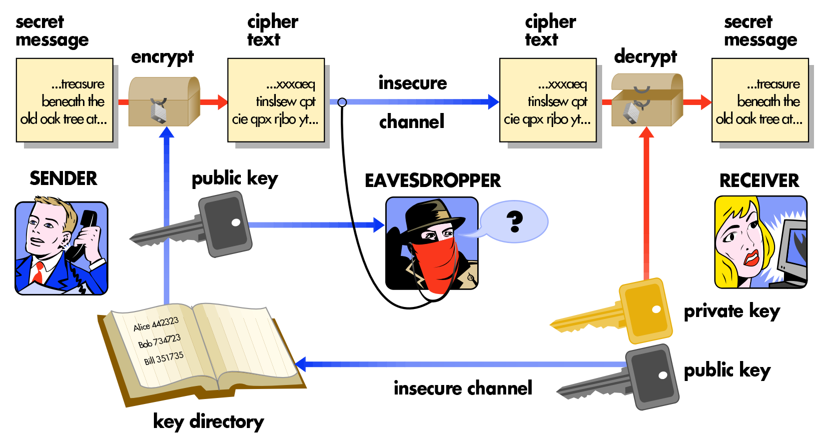
Idea of Public-Key Cryptography

Every user has *two* keys:

- Encryption key is public (so everyone can encrypt messages)
- Decryption key is only known to the receiver

Deducing the decryption key from the encryption key should be computationally infeasible.

Diagram of a Public-Key Cryptosystem



Trap-door One-Way Functions

Definition 2 (Trap-door one-way function)

A function f that satisfies the following properties:

1. *Ease of Computation:* $f(x)$ is easy to compute for any x.
2. *"Trap-door Pre-image Resistance":* Given $y = f(x)$ it is computationally infeasible to determine x *unless* certain special information used in the design of f is known.
   - When this *trap-door* k is known, there exists a function g which is easy to compute such that $x = g(k, y)$.

Key to designing public-key cryptosystems: decryption key acts as a trap door for the encryption function.

Public-Key Cryptosystem

Definition 3 (Public Key Cryptosystem (PKC))

A PKC consists of a plaintext space $\mathcal{M}$, a ciphertext space $\mathcal{C}$, a *public key* space $\mathcal{K}$, and encryption functions $E_{K_1} : \mathcal{M} \to \mathcal{C}$, indexed by public keys $K_1 \in \mathcal{K}$, with the following properties:

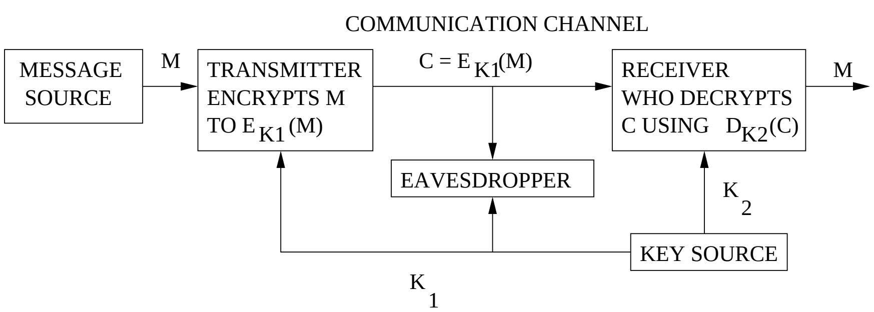
1. For every public key K_1, there exists a *private* key K_2 such that the encryption function E_{K_1} has a left inverse D_{K_2}, *i.e.*

$$D_{K_2}(E_{K_1}(M)) = M \quad \text{for all } M \in \mathcal{M}.$$

2. $E_{K_1}(M)$ and $D_{K_2}(C)$ are easy to compute when K_1 and K_2 are known.
3. Given K_1, E_{K_1}, and $C = E_{K_1}(M)$, it is computationally infeasible to find M or K_2.

By properties 1-3, E_{K_1} is a trap-door one-way function with trap door K_2.

Schematic of a Public-Key Cryptosystem



Note 1

In a public-key cryptosystem (PKC), it is *not* necessary for the key channel to be secure.

Properties of a PKC

Unlike conventional cryptosystems, messages encrypted using public key cryptosystems contain sufficient information to uniquely determine the plaintext and the key (given enough ciphertext, resources etc)

- The entropy contained in these systems is *zero*.
- This is the exact opposite of a perfectly secret system like the one-time pad.

Security in a public key cryptosystem lies solely in the computational cost of computing the plaintext and/or private key from the ciphertext (computional security).

Hybrid Encryption

All PKCs in use today are much slower (by a factor of 1000-1500 or so) than conventional systems like AES, so they are generally not used for bulk encryption. Most common uses:

- Encryption and transmission of keys for conventional cryptosystems (*hybrid* encryption) – alternative to Diffie-Hellman
- Authentication and non-repudiation via digital signatures (later).

The RSA Cryptosystem

Named after Ronald Rivest, Adi Shamir and Leonard Adleman, 1978.

Initially, NSA pressured the RSA inventors to keep their invention secret.

Independently invented in 1973 by Clifford Cocks of CESG (Communications-Electronics Security Group, part of GCHQ) after he learned about Ellis' concept of non-secret encryption

- Disclosed in 1997; see `http://cryptocellar.org/cesg/notense.pdf`

Both encryption and decryption are modular exponentiations (same modulus, different exponents):

- Encryption: $C \equiv M^e \pmod{n}$
- Decryption: $M \equiv C^d \pmod{n}$

RSA Setup

The designer

1. Selects two distinct large primes p and q (each around $2^{1536} \approx 10^{463}$)
2. Computes $n = pq$ and $\phi(n) = (p-1)(q-1)$.
3. Selects a random integer $e \in \mathbb{Z}^*_{\phi(n)}$ (so $1 \leq e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$).
4. Solves the linear congruence
$$de \equiv 1 \pmod{\phi(n)}$$
for $d \in \mathbb{Z}^*_{\phi(n)}$.
5. Keeps $d, p, q, \phi(n)$ secret and makes n and e public:
   - The public key is $K_1 = (e, n)$
   - The private key is $K_2 = \{d\}$

RSA Encryption and Decryption

Encryption: Messages for the designer are integers in $\mathbb{Z}_n^*$

- E.g. divide a bit string into blocks of bit length $\leq \lfloor \log_2(n) \rfloor$.
- Interpret each block as an integer M with $0 < M < n$ via its binary representation.

To send M encrypted, compute and send
$$C \equiv M^e \pmod{n} \text{ where } 0 < C < n \,.$$

Decryption: To decrypt C, the designer computes
$$M \equiv C^d \pmod{n} \text{ where } 0 < M < n \,.$$

RSA Toy Example

Choose $p = 11$ and $q = 19$.

$n = 11 \cdot 19 = 209$ and $\phi(n) = (11-1)(19-1) = 10 \cdot 18 = 180$.

Chose $e = 7$ and note that $\gcd(7, 180) = 1$. Then $d = 103$.
(Verify that $7 \cdot 103 \equiv 1 \pmod{180}$.)

- Public key: $(7, 209)$
- Private key: 103

Encryption of $M = 176$ is $176^7 \equiv 187 \pmod{209}$.

Decryption of $C = 187$ is $187^{103} \equiv 176 \equiv M \pmod{209}$.

(Use binary exponentiation for encryption and decryption.)

Proof of Correctness of RSA

Why do RSA encryption and decryption work?

Euler's Theorem

If $a, n \in \mathbb{Z}$ with $n > 0$ and $\gcd(a, n) = 1$, then $a^{\phi(n)} \equiv 1 \pmod{n}$.

We have
$$C^d \equiv (M^e)^d \equiv M^{ed} \pmod{n},$$
Since d is chosen such that $ed \equiv 1 \pmod{\phi(n)}$ we have
$$ed = 1 + k\phi(n) \text{ for some } k \in \mathbb{Z},$$
and hence
$$C^d \equiv M^{ed} \equiv M^{1+k\phi(n)} \equiv M \cdot M^{k\phi(n)} \equiv M(M^{\phi(n)})^k \pmod{n} .$$
Euler's Theorem implies that $M^{\phi(n)} \equiv 1 \pmod{n}$, so we have
$$C^d \equiv M(M^{\phi(n)})^k \equiv M(1)^k \equiv M \pmod{n} .$$

What if $\gcd(M, n) \neq 1$?

We have assumed that $\gcd(M, n) = 1$ in the description of RSA and for applying Euler's Theorem. Is this a problem?

- Can prove that encryption/decryption still work.
- The probability that $\gcd(M, n) \neq 1$ is $1/p + 1/q$, i.e. *very* small.
- Note that since $n = pq$ and $M < n$, $\gcd(M, n) \in \{1, p, q\}$. In the rare case that this gcd exceeds 1, we will find a factor of n.

Modular Inverses

In RSA, given $\phi(n) = (p-1)(q-1)$ and $e \in \mathbb{Z}^*_{\phi(n)}$, the designer must find $d \in \mathbb{Z}^*_{\Phi(n)}$ such that
$$ed \equiv 1 \pmod{\phi(n)} .$$

This is a particular instance of the *modular inverse* problem: given $m \in \mathbb{N}$ and $a \in \mathbb{Z}^*_m$, solve (efficiently) the congruence
$$ax \equiv 1 \pmod{m}$$
for x.

Note that this congruence is equivalent to the assertion that m divides $ax - 1$, i.e. there exists $y \in \mathbb{Z}$ such that $ax - 1 = ym$. Equivalently:

Bezout's Identity: $\quad ax - my = 1 = \gcd(a, m)$.

Linear Diophantine Equations

Given $a, b \in \mathbb{Z}$, not both 0, solve the *linear Diophantine equation*

$$ax + by = \gcd(a, b) \ .$$

Note: we may restrict to the case when $a, b > 0$:

- We have $\gcd(a, b) = \gcd(-a, b) = \gcd(a, -b) = \gcd(-a, -b)$.
- If $a < 0$, use $-a$ and solve for $(-x, y)$; similarly for $b < 0$.
- If $a = 0$ (and $b > 0$), the equation becomes

$$by = \gcd(b, 0) = b$$

with solution $y = 1$ and x can be any integer; similarly for $b = 0$.

Euclidean Algorithm

Diophantine equations and the Euclidean algorithm are named after Diophantus and Euclid, respectively. Both were Greek mathematicians who lived in Alexandria around 300 BCE.

The Euclidean algorithm finds greatest common divisors via repeated division with remainder. Given $a, b \in \mathbb{Z}$, $b > 0$, and $\gcd(a, b) = 1$:

$$\begin{array}{rll}
a = q_0 b + r_0 & & q_0 = \lfloor a/b \rfloor, 0 < r_0 < b \\
b = q_1 r_0 + r_1 & & q_1 = \lfloor b/r_0 \rfloor, 0 < r_1 < r_0 \\
r_0 = q_2 r_1 + r_2 & & q_2 = \lfloor r_0/r_1 \rfloor, 0 < r_2 < r_1 \\
\vdots & & \\
r_{n-3} = q_{n-1} r_{n-2} + r_{n-1} & & r_{n-1} = \gcd(a, b) \\
r_{n-2} = q_n r_{n-1} + r_n & & r_n = 0
\end{array}$$

Termination

Notice that the sequence of remainders (the r_i) is non-negative and strictly decreasing

- Thus, the sequence is finite (algorithm terminates).

Theorem 1 (Lamé, 1844)

$n < 5 \log_{10} \min(a, b)$.

More exactly, Lamé's Theorem states

$$n \leq \log_\tau(\min(a, b) + 1)$$

where $\tau = (1 + \sqrt{5})/2$ is the golden ratio.

Extended Euclidean Algorithm Via Back Substitution

$$\begin{array}{rcrcll}
\gcd(a, b) & = & r_{n-1} & = & r_{n-3} - q_{n-1} r_{n-2} & (1) \\
& & r_{n-2} & = & r_{n-4} - q_{n-2} r_{n-3} & (2) \\
& & r_{n-3} & = & r_{n-5} - q_{n-3} r_{n-4} & (3) \\
& & \vdots & & \vdots &
\end{array}$$

So
$$\begin{aligned}
\gcd(a, b) &\overset{(1)}{=} r_{n-3} + (-q_{n-1}) r_{n-2} \\
&\overset{(2)}{=} r_{n-3} + (-q_{n-1})(r_{n-4} - q_{n-2} r_{n-3}) \\
&= (-q_{n-1}) r_{n-4} + (1 + q_{n-1} q_{n-2}) r_{n-3} \\
&\overset{(3)}{=} (-q_{n-1}) r_{n-4} + (1 + q_{n-1} q_{n-2})(r_{n-5} - q_{n-3} r_{n-4}) \\
&= (\cdots) r_{n-5} + (\cdots) r_{n-4} \\
&= \cdots \\
&= (\cdots) a + (\cdots) b = xa + yb \ .
\end{aligned}$$

Extended Euclidean Algorithm Via Bezout's Method

Let $A_{-2} = 0$, $A_{-1} = 1$, $B_{-2} = 1$, $B_{-1} = 0$ and

$$A_k = q_k A_{k-1} + A_{k-2}, \qquad B_k = q_k B_{k-1} + B_{k-2}$$

for $k = 0, 1, \ldots$.

We have $A_n = a$ and $B_n = b$ (n from above), and

$$A_k B_{k-1} - B_k A_{k-1} = (-1)^{k-1} \ .$$

Putting $k = n$ yields

$$A_n B_{n-1} - B_n A_{n-1} = (-1)^{n-1}$$
$$a(-1)^{n-1} B_{n-1} + b(-1)^n A_{n-1} = 1 \ .$$

Thus, a solution of $ax + by = 1$ is given by

$$x = (-1)^{n-1} B_{n-1}, \quad y = (-1)^n A_{n-1} \ .$$

Bezout Tableau

Bezout's method can be represented compactly as a tableau as follows:

		q_0	q_1	q_2	$\cdots$	q_{n-1}
0	1	A_0	A_1	A_2	$\cdots$	A_{n-1}
1	0	B_0	B_1	B_2	$\cdots$	B_{n-1}

Each entry in the second and third row is computed by multiplying the quotient q_i in the current column by the previous entry in the same row and adding it to the entry before that in the same row.

Example 4

Solve $44x + 13y = 1$.

$$44 = 3 \cdot 13 + 5$$
$$13 = 2 \cdot 5 + 3$$
$$5 = 1 \cdot 3 + 2$$
$$3 = 1 \cdot 2 + 1$$

		3	2	1	1
0	1	3	7	10	17
1	0	1	2	3	5

$$x = (-1)^{4-1} \cdot 5 = -5$$
$$y = (-1)^4 \cdot 17 = 17$$

Modular Inverses

Recall that $\mathbb{Z}_m^* = \{a \in \mathbb{Z}_m \mid \gcd(a, m) = 1\}$ is the set of integers between 1 and m that are coprime to m.

$\mathbb{Z}_m^*$ consists of exactly those integers that have *modular inverses*:

- for every $a \in \mathbb{Z}_m^*$, there exists $x \in \mathbb{Z}_m^*$ such that $ax \equiv 1 \pmod{m}$.

Computing Modular Inverses

Given $a \in \mathbb{Z}_m^*$, solve the linear congruence $ax \equiv 1 \pmod{m}$ for $x \in \mathbb{Z}_m^*$.

- We want x such that

  $$m \mid ax - 1 \iff ax - 1 = ym \iff ax - my = 1$$

  for some $y \in \mathbb{Z}$.
- Can be solved using the Extended Euclidean Algorithm.
- We only need to compute the B_i because we only need x, not y.
- When using Bezout's method, be sure to start with $a = q_0 m + r_0$, even if $a < m$ (so $q_0 = 0$), else you get the wrong count for the number n of division steps.
- Always check your answer!