

# Security and Privacy, Blatt 1

Franziska Hutter (3295896)  
Felix Truger (3331705)  
Felix Bühler (2973410)

5. Juni 2018

## Problem 1: Matching Algorithm

The desired algorithm is listed below (pseudo code):

```
struct result {
    bool match
    string matcher
}

result match(m, t)
    result res = {true, ""}

    if (m is encrypted && t is encrypted) // both encrypted
        if (m is encrypted_symmetrically && t is encrypted_symmetrically)
            k_m = encryption_key of m
            k_t = encryption_key of t
            keys_match = match(k_m, k_t)
            res.match = keys_match.match
            res.matcher += keys_match.matcher
            t = apply_matcher(t, res.matcher) // apply the matcher to whole term
            if (keys_match) // both encrypted under same method and key
                inner_m = decrypt(m) // retrieve the plaintext of m
                inner_t = decrypt(t) // retrieve the plaintext of t
                inner_match = match(inner_m, inner_t) // match the plaintexts
                res.match = inner_match.match
                res.matcher += inner_match.matcher
            else
                res.match = false // no matcher for the encryption keys
        else if (m is encrypted_asymmetrically && t is encrypted_asymmetrically)
            k_m = encryption_key of m
            k_t = encryption_key of t
            if (k_m == k_t)
                inner_m = decrypt(m) // retrieve the plaintext of m
                inner_t = decrypt(t) // retrieve the plaintext of t
                inner_match = match(inner_m, inner_t) // match the plaintexts
                res.match = inner_match.match
                res.matcher += inner_match.matcher
            else
                res.match = false // different keys
        else
            res.match = false // different encryption methods
    else
        if (m is encrypted || t is encrypted)
            res.match = false // one encrypted one not
        else // both unencrypted
            if (m is tuple && t is tuple)
                // both are tuples, so check for match component-wise (assuming two components)
                first_match = match(first_component(m), first_component(t))
                t = apply_matcher(t, first_match.matcher) // apply matcher to whole term
                second_match = match(second_component(m), second_component(t))
                t = apply_matcher(t, second_match.matcher) // apply matcher to whole term
                res.match = first_match && second_match
                res.matcher += first_match.matcher + second_match.matcher
            else
                if (m is tuple || t is tuple)
                    res.match = false // one is tuple one not
                else
                    // both neither tuples nor encrypted, so m is constant and t is constant or variable
                    if (t is variable)
                        res.matcher += "ground_substitution:_" + t + " ->_" + m + "]\n"
                        // t must be substituted by m
                    else
                        if (!equals(m, t))
                            res.match = false // constants mismatch

    if (res.match)
        print "matcher:_" + res.matcher
    else
        res.matcher = NULL
    return res

term apply_matcher(t, matcher)
    foreach(substitution in matcher)
        foreach(symbol in t)
            if (symbol = substitution.symbol)
                replace(symbol, substitution.substitute)
    return t
```

For simplicity we assume  $n$  in the following as the maximum of the amount of symbols in  $m$  and  $t$ .

Time complexity (Master Theorem):

$$f(n) = a \cdot f\left(\frac{n}{b}\right) + c(n)$$

$$a = 2$$

$$b = 2$$

$$c(n) \in \mathcal{O}(n^d); d = 2$$

$$b^d = 2^2 = 4 > a \implies f(n) \in \mathcal{O}(n^2)$$

Space complexity:

$$f(n) \in \mathcal{O}(n)$$

## Problem 2: Basics - Probability Theory

—

## Problem 3: Basics - Algorithms

a)

Product space:  $\Omega_A^{prod} = \{0, 1\} \times M \times \{0, 1\}^t \times \{0, 1\}^2$

Probability space:  $(\Omega_A^{prod}, 2^{\Omega_A^{prod}}, P)$

b)

$$Pr[A(z) = 1] = Pr[c = 1] \cdot (Pr[a = 1] + Pr[b \cdot a = 1]) = \frac{1}{2} \cdot \frac{1}{15} = \frac{1}{30}$$

c)

$$Pr[d \neq \perp] = Pr[c = 1] \cdot Pr[a < 12] = \frac{1}{2} \cdot \frac{12}{15} = \frac{2}{5}$$

(In words: the probability that  $d$  is assigned in a run of  $A$ .)

d)

$$Pr[A(z) \leq 24 | b = 2] = Pr[a = 12] = \frac{1}{15}$$

## Problem 4: Basics - Group Theory

a)

- $(\mathbb{Z}_8^*, \cdot_8)$ : Nein, da es isomorph zu  $\mathbb{Z}_2^* * \mathbb{Z}_2^*$  ist.  
(→ Es besitzt keine Primitivwurzel.)
- $(\mathbb{Z}_{10}^*, \cdot_{10})$ : Ja. Generator ist 3 oder 7.

Generator = x	3	7
$x^0$	1	1
$x^1$	3	7
$x^2$	9	9
$x^3$	7	3
$x^4$	1	1

b)

Aus der Vorlesung:

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \gcd(a, n) = 1\} \rightarrow \mathbb{Z}_n^* = \{a, b \in \mathbb{Z}_n \mid \gcd(a * b, n) = 1\}$$

Multiplikation ist ein Gesetz der Komposition auf  $\mathbb{Z}_n^*$ .

$$a, b, c \in \mathbb{Z}_n^*$$

- Die Multiplikation ist assoziativ auf  $\mathbb{Z}_n^*$ :  $(a * b) * c = abc = a * (b * c)$   
( $\gcd((a * b) * c, n) = 1 = \gcd(a * b * c, n) = \gcd(a * (b * c), n)$ )
- Ebenso ist die Multiplikation kommutativ:  $a * b = b * a$   
( $\gcd(a * b, n) = 1 = \gcd(b * a, n)$ )

- Neutrales Element:

Wir nehmen als Identität 1. Natürlich ist,  $\forall x \in \mathbb{Z} : \gcd(1, x) = 1$ , also  $1 \in \mathbb{Z}_n^*$ . Dann  $a * 1 = a = 1 * a$ . Somit erfüllt 1 die Eigenschaft des neutralen Elements.

- Inverses Element:

$\forall x \in \mathbb{Z} : ax \equiv 1 \pmod{n}$ . Es existiert genau dann, wenn  $a$  teilerfremd zu  $n$  ist, weil in diesem Fall  $\gcd(a, n) = 1$ . Und nach Bezous existiert somit ein Inverses Element.