DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Implementation of Attribute-Based Encryption in Rust on ARM Cortex M Processors

Daniel Bücheler

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Implementation of Attribute-Based Encryption in Rust on ARM Cortex M Processors

Implementierung von Attributbasierter Verschlüsselung in Rust auf ARM Cortex M Prozessoren

Author:	Daniel Bücheler
Supervisor:	Prof. Dr. Claudia Eckert
Advisor:	Stefan Hristozov
Submission Date:	15.04.2021

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.04.2021 Daniel Bücheler

Acknowledgments

Abstract

Contents

1 Introduction

1.1 Section

Citation test [**latex**].

1.1.1 Subsection

See Table 1.1, Figure 1.1, Figure 1.2, Figure 1.3.

Table 1.1: An example for a simple table.

A	B	C	D
1	2	1	2
2	3	2	3

$$\begin{tikzcd}
R_1 \arrow[r, dash] \arrow[d, dash] & R_2 \arrow[r, dash] \arrow[d, dash] & R_5 \\
R_3 \arrow[r, dash] & R_4 &
\end{tikzcd}$$

Figure 1.1: An example for a simple drawing.

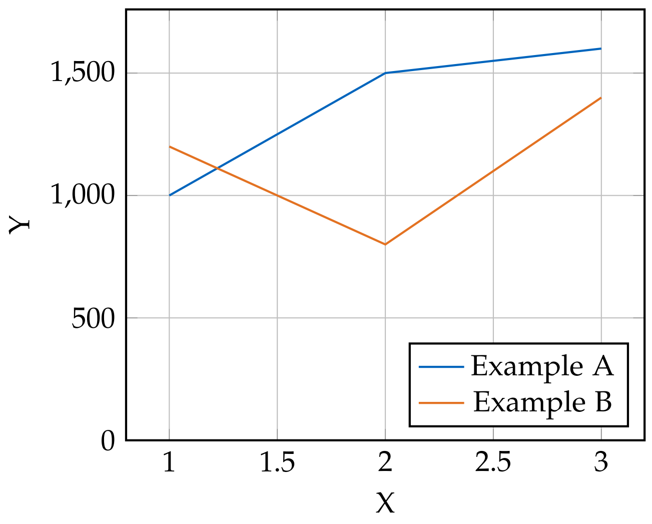


Figure 1.2: An example for a simple plot.

```
SELECT * FROM tbl WHERE tbl.str = "str"
```

Figure 1.3: An example for a source code listing.

2 Background

This chapter shall provide a high-level introduction to the cryptographical and mathematical tools used to implement attribute-based encryption in this thesis. For further reference, please refer to

2.1 Classic Symmetric and Asymmetric Cryptography

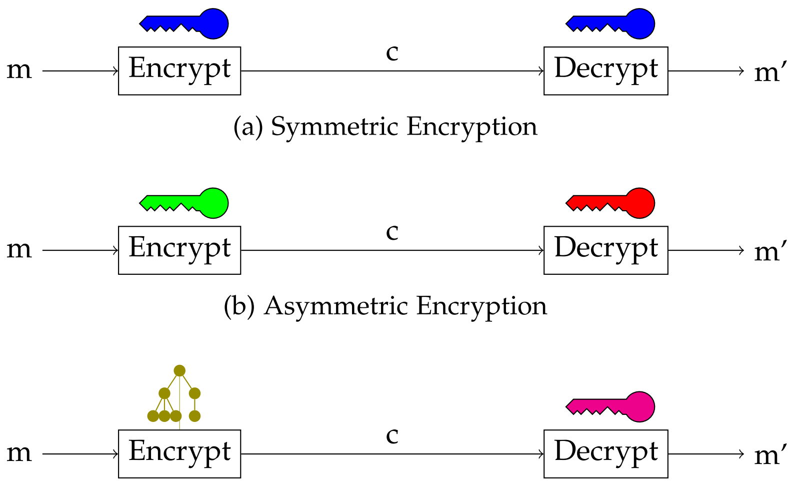
Today's conventional cryptography knows two main classes of cryptosystems: *Symmetric* or *Private-Key systems* and *Asymmetric* or *Public-Key systems*. The main difference lies in their use of encryption and decryption keys:

In *symmetric* systems, the key used for encryption and decryption is identical. That is, a user *Alice* encrypting a message to send to another user *Bob* will encrypt the message using a key k that she had agreed on with Bob beforehand. When he receives the encrypted message, Bob will use the same key k to decrypt it. See Figure 2.1a.

In *asymmetric* systems, on the other hand, the keys used for encryption and decryption differ. When Alice encrypts a message with key k_{enc}, she will not be able to encrypt it again. Instead, when Bob receives the encrypted message, he will use a different key k_{dec} to decrypt it. See Figure 2.1b. Thus, in symmetric systems, keys always come in *pairs* of an *encryption key k_{enc}* and a *decryption key k_{dec}*. Because k_{enc} can not be used to decrypt messages meant for Bob, there is no harm to making it publicly available. For example, he might put it up on his website for anyone wishing to send him an encrypted message to download. This is why k_{enc} is also often called the *public key* and k_{dec} the *private key*.

Asymmetric cryptosystems make secure communication among a large group of participants much easier: Consider n participants wanting to communicate securely using a symmetric system. Each participant would need to share a unique secret key with each of the other participants, requiring a total of $\frac{n(n-1)}{2}$ keys. In the asymmetric setting, one key per participant is sufficient: The same public key may be shared with the whole group, as the private key remains private anyway. This reduces the total number of keys to n.

Another problem remains, however: Encrypting a single message to a large number of participants requires encrypting it with everyone's public key separately. For a large



(a) Symmetric Encryption

(b) Asymmetric Encryption

(c) Attribute-Based encryption

Figure 2.1: Keys used for encryption and decryption in different classes of encryption schemes

number of recipients, this is inefficient. So, for example, to encrypt a message for all students of a certain university, we'd need to obtain each student's public key and encrypt the message with each key separately.

Even worse, what if we want to encrypt data for any student of said university, even if they *haven't joined the university yet*. In this case, our only option using classic public-key cryptography would be to have some trusted instance encrypt the data for any new student after they joined the university. Attribute-Based encryption solves this problem much more nicely.

2.2 Attribute-Based Encryption

Attribute-Based Encryption Schemes (ABE schemes) are asymmetric in the sense that different keys are used for encryption and decryption. However, in constrast to classic asymmetric systems, the 'public key' used for encryption is not attached to an identity, but to certain attributes defined by the system. This is represented by a tree of attributes in Figure 2.1c.

2.2.1 KP-ABE and CP-ABE

ABE schemes can further be divided into *Key-Policy* and *Ciphertext-Policy* schemes (KP-ABE and CP-ABE, respectively). In KP-ABE, the ciphertext is associated with a set of attributes, and the key is associated with an access structure. CP-ABE works the other way around, so the key is associated with a set of attributes and the ciphertext is encrypted under an access structure.

In both cases, a ciphertext can be decrypted if and only if the set of attributes specified in one part satisfy the access policy associated with the other part.

2.2.2 Components of an ABE scheme

With respect to ABE, we say that a ciphertext is *encrypted under an access structure* (*under a set of attributes*, respectively) if it has been encrypted using the public key derived from the access structure (set of attributes, resp.). Because the public key w.r.t. some access structure (set of attributes, resp.) can be derived by anyone, private keys are usually referred to as just *keys*.

Definition 2.1. An Attribute-Based Encryption scheme (ABE scheme) consists of the following four algorithms: [1]

- *Setup*. Run once by the central trusted authority (key generation center). Sets up the system and outputs a public master key PK and a private master secret s.
- *KeyGen(PK, s, ω)*. Run exclusively by the trusted authority. For a given set of attributes (CP-ABE) or access structure (KP-ABE) ω, uses the master secret s to generate a private key k corresponding to the given access structure.
- *Encrypt(PK, m, ω)*. Run by any participant of the system. On input of a plaintext message m and an access structure (CP-ABE) or set of attributes (KP-ABE) ω, outputs a ciphertext c
- *Decrypt(c, k)*. Run by any participant holding a private key generated by *KeyGen*. On input of a ciphertext c created by *Encrypt* and a key k generated by *KeyGen*, it outputs a correctly decrypted message if and only if
  - in the case of CP-ABE: the access structure under which m was encrypted is satisfied by the set of attributes under which k was created
  - in the case of KP-ABE: the set of attributes under which m was encrypted satisfies the access structure under which k was created.

How exactly these algorithms work in concrete ABE schemes will be discussed in Section 2.5.

2.2.3 Access Trees

Of course, we need a way to define the policies or access structures associated with the key (KP-ABE) or the ciphertext (CP-ABE), respectively. We will be using the definition of *Access Trees* from [1]. Each leaf of this tree is labelled with an attribute, and each interior node is labelled with an integer, the threshold for it to be satisfied [1].

Definition 2.2. Access Tree [1].
An interal note x of an access tree is defined by its children and a threshold value d_x. If x has num_x children, then its threshold value satisfies $0 < d_x \leq num_x$.
A leaf node x is defined by an attribute and a threshold value $k_x = 1$.

[1] also defines the following functions for working with access trees: The parent of a node x in the access tree is denoted by $\text{parent}(x)$. If x is a leaf node, $\text{att}(x)$ denotes the attribute associated with x; otherwise it is undefined. The children of a node x are numbered from 1 to num_x. Then $\text{index}(y)$ denotes the unique index of y among the children of its parent node.

Definition 2.3. Satisfying Access Trees [1].
Let $\mathcal{T}$ be an access tree with root r and $\mathcal{T}_x$ the subtree with x as its root. If a set of attributes γ satisfies the access tree $\mathcal{T}_x$, we write $\mathcal{T}_x(\gamma) = 1$; otherwise $\mathcal{T}_x(\gamma) = 0$.
If x is a leaf node, then $\mathcal{T}_x(\gamma) = 1$ if and only if $\text{attr}(x) \in \gamma$.
If x is an internal node, then $\mathcal{T}_x = 1$ if and only if d_x or more of the children x' of x return $\mathcal{T}_{x'}(\gamma) = 1$.

Using this construction with threshold gates, we can express *A AND B* as a node with two children *A* and *B* and threshold 2, and express *A OR B* as a node with two children *A* and *B* and threshold 1 [2].

2.3 Shamir's Secret Sharing

This secret sharing scheme based on polynomial interpolation was first introduced by Adi Shamir in 1979 [3]. It allows a secret s, which is generally just a number, to be shared among a number of n participants. The shares are computed such that s can be reconstructed if, and only if, at least k participants meet and combine their shares. Such a theme is then called a (k, n)-threshold scheme. [3]

2.3.1 Lagrange interpolation

Shamir's scheme makes use of a property of polynomials: A polynomial of degree d is unambiguously determined by $d + 1$ points (x_i, y_i). In other words, any polynomial of degree d can be unambiguously interpolated (reconstructed) from $d + 1$ distinct points.

To interpolate a polynomial of degree d from $d+1$ given points $(x_1, y_1), \dots, (x_{d+1}, y_{d+1})$, we can make use of the lagrange basis polynomials: [2]

Definition 2.4. Lagrange interpolation: Given a set of $d+1$ points $(x_1, y_1), \dots, (x_{d+1}, y_{d+1})$.
Then the polynomial

$$L(x) = \sum_{k=0}^{d} l_{\omega,x_k}(x) \cdot y_k \tag{2.1}$$

is the lagrange interpolation polynomial for that set of points, where $\omega = \{x_1, \dots, x_{d+1}\}$ and $l_{\omega,k}(x)$ are the Lagrange basis polynomials:

$$l_{\omega,k}(x) = \prod_{\substack{i\in\omega \\ i\neq k}}^{d} \frac{x-i}{k-i} \tag{2.2}$$

This polynomial has degree d. If the points (x_i, y_i) lie on a d-degree polynomial, then the lagrange interpolation $L(x)$ is *exactly* that polynomial.

On the other hand, if there are less than $d+1$ points of a d-degree polynomial known, there are infinitely many d-degree polynomials that pass through all given points. [3]

2.3.2 Secret sharing with polynomials

To share our secret, we now hide it in a polynomial and give out points on this polynomial as secret shares. Using the lagrange basis polynomials, we can then reconstruct $p(x)$ and thus the secret if we know enough shares. [3]

Definition 2.5. Shamir's (k, n)-threshold secret sharing scheme [3]. To share a secret s among n participants such that s can be recovered if and only if k or more shares are combined, do:

1. Pick coefficients $a_1, ..., a_{k-1}$ at random
2. Set $a_0 = s$. This results in the polynomial $p(x) = a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}$. Note that $p(0) = s$.
3. The secret shares are $(1, p(1)), (2, p(2)), \dots, (n, p(n))$. Give one to each participant.

To reconstruct the secret from any subset of k shares, interpolate the polynomial $p(x)$ and evaluate $p(0) = s$.

See also Figure 2.2 for illustration. In practice, the numbers would be far bigger and calculations wouldn't be performed over the real numbers, but rather a finite field modulo a prime. [3]

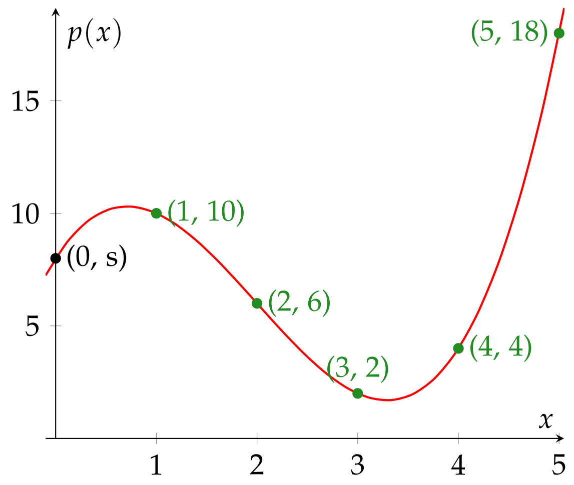


Figure 2.2: Example for a $(5,4)$-threshold scheme with $s = 8$ and $p(x) = 8 + 7x - 6x^2 + x^3$. The five green-colored points are distributed as the secret shares. As $p(x)$ has degree three, at least four shares are required to reconstruct s.

2.3.3 Secret Sharing in Attribute Based Encryption

To realize an Access Tree that „gives away" a secret if and only if it is satsified by a set of attributes, we can recursively use Shamir's Secret Sharing scheme:

See Figure 2.3 for a sample of an access tree. It is satisfied by any set of attributes that contains two of A, B and either C or D. That is, $\{A, B\}$ would satsify the tree, just as $\{B, D\}$ would, but $\{C, D\}$ would not be sufficient.

Quite simplified, we now employ Shamir's Secret Sharing Scheme recursively on each internal node of the Access Tree: For a node x with threshold d_x and num_x children, we define a (d_x, num_x)-threshold scheme and embed one share of the secret in each child. Begin in the root, and set s as the secret we want to embed in the tree. For all other nodes, set s as the secret share received from the parent node.

If the child is a leaf, we modify the share such that it can only be used if the relevant attribute is present (we will not discuss how this is done at this point).

Now, let ω be a set of attributes. We have built our tree in such a way that the share embedded in a leaf node u can be used only if $\text{attr}(u) \in \omega$. That means, a leaf node's secret share can be used if and only if the set of attributes satisfies this leaf node.

For the internal nodes x, the use of a (d_x, num_x)-threshold scheme ensures that the secret embedded in x can be reconstructed if and only if the secret shares of at least d_x child nodes can be used, i.e. at least d_x child nodes are satisfied. Following this recursive definition up to the root, we can see that our secret s embedded in the root

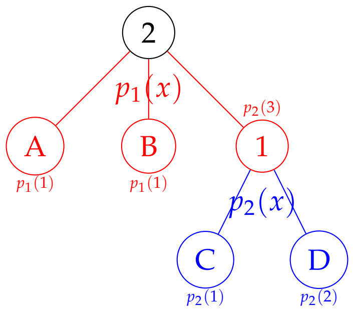


Figure 2.3: Sample Access Tree to show how Shamir's Secret Sharing is used to embed a secret in the tree. $p_1(x)$ is a the polynomial of a $(2,3)$-threshold scheme, $p_2(x)$ of a $(1,2)$-threshold scheme. Shown in small are the secret embedded into each node.

can be reconstructed exactly if ω satisfies the Acess Tree.

2.4 Elliptic Curves

The mathematics of modern cryptosystems (including, but not limited to ABE) work on a great variety of mathematical structures, and elliptic curves are just one of them. They have become very popular since their discovery by *citation needed* in *year needed* because they provide an equivalent security level at shorter key length and smaller computational cost than other systems (e.g. based on RSA) [4]. Exact definitions and notations differ, these are taken from the textbook *Introduction to Modern Cryptogaphy* by Katz and Lindell [4].

Definition 2.6. Given a prime $p \geq 5$ and curve parameters $a, b \in \mathbb{Z}_p$ with $4a^2 + 27b^2 \neq 0 \bmod p$, the Elliptic Curve over $\mathbb{Z}_p$ is: [4]

$$E(\mathbb{Z}_p) := \{(x,y) \mid x, y \in \mathbb{Z}_p \text{ and } y^2 = x^3 + ax + b \bmod b\} \cup \{\mathcal{O}\} \tag{2.3}$$

a and b are called the curve parameters, and the requirement that $4a^2 + 27b^2 \neq 0 \bmod p$ makes sure that the curve has no repeated roots. The curve is simply the set of points $(x,y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ that satisfy the curve equation $y^2 = x^3 + ax + b \bmod b$. One special point is added, the *point at infinity* denoted by $\mathcal{O}$. This will help define the point addition as a group operation in the next paragraph. [4]

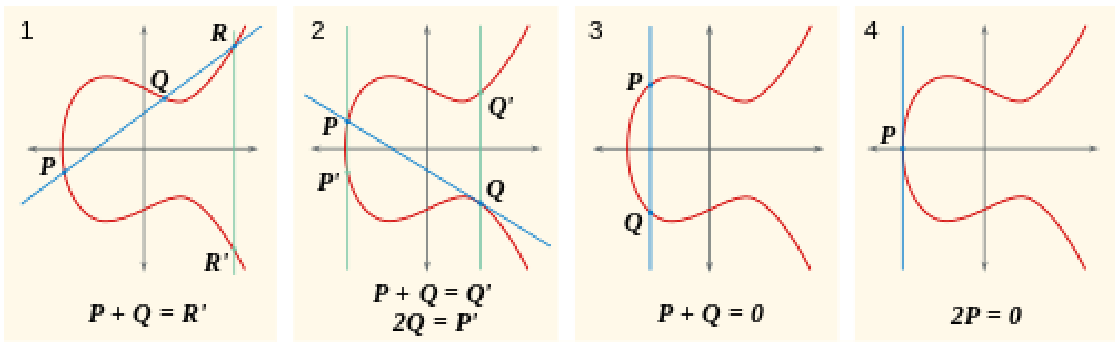


Figure 2.4: Elliptic Curve point addition
(Image by SuperManu, licensed under Creative Commons.)

2.4.1 Point Addition

Now, it is possible to show that every line intersecting a curve $E(\mathbb{Z}_p)$ intersects it in exactly three points, if you (1) count tangential intersections double and (2) count any vertical line as intersecting the curve in the point at infinity $\mathcal{O}$. Therefore, $\mathcal{O}$ can be thought of as sitting "above" the end of the y-axis [4]. Figure 2.4 shows all four different combinations, feel free to convince yourself that this statement indeed makes sense for the plotted curve.

Using this intersecting line, we can define an operation on curve points.

Definition 2.7. Given an Elliptic Curve $E(\mathbb{Z}_p)$, we define a binary operation called *(point) addition* and denoted by $+$: [4]
Let $P_1, P_2 \in E(\mathbb{Z}_p)$.

- For two points $P_1, P_2 \neq \mathcal{O}$ and $P_1 \neq P_2$, their sum $P_1 + P_2$ is evaluated by drawing the line through P_1 and P_2 (if $P_1 = P_2$, draw a tangential line). This line will intersect the curve in a third point, $P_3 = (x_3, y_3)$. Then the result of the addition is $P_1 + P_2 = (x_3, -y_3)$, i.e. P_3 is reflected in the x-axis. If $P_3 = \mathcal{O}$, then the result of the addition is $\mathcal{O}$.
- If $P_1, P_2 \neq \mathcal{O}$ and $P_1 = P_2$, as above but draw the line as tangent on the curve in P_1
- If $P_1 = \mathcal{O}$, then $P_1 + P_2 = P_2$ and vice-versa.

We will be adding points to themselves a lot. Therefore, we define for ease of notation:

Definition 2.8. Point-Scalar multiplication: Given a point $P \in E(\mathbb{Z}_p)$ and a scalar $d \in \mathbb{N}$:

$$d \cdot P = \underbrace{P + P + \cdots + P}_{d \text{ times}} \tag{2.4}$$

That is, the product of a scalar with a point is again a point on our curve.

2.4.2 Groups on Elliptic Curves

Theorem 2.1. *The points of an Elliptic Curve $E(\mathbb{Z}_p)$ plus the addition law as stated in Definition 2.7 forms an abelian (commutative group) [4], [5]:*

Proof. A formal proof is outside the scope of this thesis, but here's some informal reasoning about the group axioms:

- Existence of Identity: $P + \mathcal{O} = P$ (as per definition)
- Commutativity: For all $P_1, P_2 \in E(\mathbb{Z}_p)$, $P_1 + P_2 = P_2 + P_1$ (obvious, because the line through P_1 and P_2 will be the same)
- Unique inverse: For any point $P = (x, y) \in E(\mathbb{Z}_p)$, the unique inverse is $-P = (x, -y)$ (obvious).
- Associativity: For all $P_1, P_2, P_3 \in E(\mathbb{Z}_p)$, $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$ (much less obvious, see e.g. [5, Chapter 2.4] for a proof).

□

Of particular interest to cryptography are *cyclic* groups on elliptic curves:

Definition 2.9. A (multiplicative) group $\mathbb{G}$ is cyclic if there is an element $g \in \mathbb{G}$ that generates $\mathbb{G}$, i.e. $\mathbb{G} = \langle g \rangle = \{g^n | n \in \mathbb{Z}\}$.

Translated to our (additive) groups on elliptic curves, this means that there is a generator point $P \in E(\mathbb{Z}_p)$, such that every point $Q \in E(\mathbb{Z}_p)$ can be written as $Q = nP$ with some $n \in \mathbb{N}$.

Theorem 2.2. *[4] Let $\mathbb{G}$ be a finite group of order n, i.e. $|\mathbb{G}| = n$. Let $g \in \mathbb{G}$ be an element of $\mathbb{G}$ with order k, i.e. $k = |\langle g \rangle|$*

Then $k | n$, i.e. the order of g divides the group order n.

Proof. See [4, Proposition 8.54]. □

There is an important consequence to this fact: If a group has prime order, all points except the identity are generators. This stems from the fact that a prime number has exactly two divisors: One (the order of the identity) and itself (the order of all other points).
This follows from the fact that for any point $P \in E(\mathbb{Z}_p)$, its order $\text{ord}(P) = |\langle P \rangle|$ must divide the group order. A prime has exactly two divisors: One (the order of $\mathcal{O}$) and itself (the order of all other points).

Again, translated to Elliptic Curves this means that if the number of points $\#E(\mathbb{Z}_p)$ on a curve is prime, all points except $\mathcal{O}$ are generators. These cyclic elliptic curve groups (or, cyclic subgroups of non-cyclic elliptic curves) are exactly the groups we are interested in for doing actual cryptography. See also [4, p. 321].

2.4.3 Hardness Assumptions

Most ECC schemes are build upon three hardness assumptions: The Discrete Logarithm Problem (DLP), the Decisional Diffie-Hellman Problem (DDHP) and the Computational Diffie-Hellman Problem (CDHP). Given an additive, cyclic group $\mathbb{G}$ with $P \in \mathbb{G}$ a generator, they are stated as follows:

Discrete Logarithm Problem. Given an arbitrary $Q \in \mathbb{G}$, compute an $n \in \mathbb{N}$ such that $nP = Q$.

Computational Diffie-Hellman Problem. Given the triple (P, aP, bP) where $a, b \in \mathbb{N}$ chosen uniformly at random, compute abP.

Decisional Diffie-Hellman Problem. Given two triples (aP, bP, abP) and (aP, bP, Q) where $a, b \in \mathbb{N}$ and $Q \in \mathbb{G}$ chosen uniformly at random, distinguish between the two.

Now, the hardness *assumption* is that, for some groups, these problems are hard to solve, i.e. solving them requires so much time and computational power that it is infeasible. From this, we can build secure asymmetric encryption schemes.

These three problems might all be assumed to be *hard*, but that doesn't mean they are equally so: If, in a certain group $\mathbb{G}$, the DLP problem is easy, so is CDHP: Just compute a and b, and then use them to calculate abP. And if CDHP is easy w.r.t some $\mathbb{G}$, so is DDHP: Just compute abP, and compare the third element of each tuple. The inverse is not generally true, i.e. there are groups in which DLP and CDHP are hard to solve, even though DDHP is easy to solve. In that sense, DLP is the hardest and DDHP the easiest of the three. [4] (TODO find out if this is true. [6])

2.4.4 Bilinear Pairings

Introduction

The final primitive needed for pairing-based encryption schemes are bilinear pairings. These are functions mapping two points on (possibly different) elliptic curves to elements of a finite field (*not* another point on a curve).[7]

Let $n \in \mathbb{N}_0$, $\mathbb{G}_1$ denote an additive abelian group of order n with generator P and identity $\mathcal{O}$. Let $\mathbb{G}_T$ be another group of order n, this time written multiplicatively with generator g and identity 1. A *bilinear pairing* then is a function $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ with the following properties:

- *Bilinearity.* For all $Q_1, Q_2, Q_3 \in \mathbb{G}_1$, $e(Q_1 + Q_2, Q_3) = e(Q_1, Q_3) + e(Q_2, Q_3)$ and $e(Q_1, Q_2 + Q_3) = e(Q_1, Q_2) + e(Q_1, Q_3)$
- *Non-Degeneracy.* $e(P, P) \neq 1$

Using a bilinear pairing with a group $\mathbb{G}_T$ where the DLP is easy to solve, the DLP in $\mathbb{G}_1$ can also be solved easily: To find n such that $Q = nP$, compute $e(P, Q) = e(P, nP) = e(P, \underbrace{P + \cdots + P}_{n \text{ times}}) = \underbrace{e(P, P) \cdots e(P, P)}_{n \text{ times}} = e(P, P)^n$. Thus, the discrete logarithm of Q with respect to P is the discrete logarithm of $e(P, Q)$ with respect to $g = e(P, P) \in \mathbb{G}_T$. [6]

Bilinear Hardness Assumption

Since the DLP can be easy to solve using pairings, we need to adapt our hardness assumption. Therefore, we define

The *Bilinear Diffie-Hellman Problem* (BDHP): Given P, aP, bP, cP with $a, b, c \in \{1, \ldots, n - 1\}$ chosen randomly, compute $e(P, P)^{abc}$. [6]

If the BDHP is hard for a pairing e on groups $\mathbb{G}_1$ and $\mathbb{G}_T$, this implies that the DLP is hard in both $\mathbb{G}_1$ and $\mathbb{G}_T$. [6]

2.5 Different ABE schemes for use in Embedded Devices

2.5.1 Yao, Chen and Tian 2015

This scheme was described by Yao, Chen and Tian [2] in 2015. In 2019, Tan, Yeow and Hwang [8] proposed an enhancement, fixing a flaw in the scheme and extending it to be a hierarchical KP-ABE scheme.

Yao, Chen and Tian's ABE scheme (hereafter written just YCT) is a KP-ABE scheme that does not use any bilinear pairing operations. Instead, the only operation performed on Elliptic Curves are point-scalar multiplication [2]. This makes it especially useful for our resource-constrained context, as bilinear pairings are significantly more costly in terms of computation and memory.

As opposed to other ABE schemes based on pairings, YCT uses a hybrid approach similar to Elliptic Curve Integrated Encryption Standard (ECIES): The actual encryption of the plaintext is done by a symmetric cipher, for which the key is derived from a curve point output by the YCT scheme. If a key's access policy is satisfied by a certain plaintext, this curve point and thus the symmetric encryption key can be reconstructed, allowing for decryption. [2]

The four algorithms of an ABE scheme are defined as follows:

Setup [2]. The attribute universe is defined as $\mathrm{U} = \{1, 2, \ldots, n\}$ and is fixed.

For every attribute $i \in U$, choose uniformly at random a secret number $s_i \in \mathbb{Z}_q^*$. Then the public key of attribute i is $P_i = s_i \cdot G$ (i.e. a curve point).

Also, choose uniformly at random the master private key $s \in \mathbb{Z}_q^*$, from which the master public key $PK = s \cdot G$ is derived.

Publish $Params = (PK, P_1, \ldots, P_n)$ as the public parameters, privately save $MK = (s, s_1, \ldots, s_n)$ as the private master key.

KeyGen(Γ*, MK)* [2]. Input: Access Tree Γ and master key MK.

For each node u in the Access Tree Γ, recursively define polynomials $q_u(x)$ with degree $(d_u - 1)$, starting from the root.

For the root r, set $q_r(0) = s$ and randomly choose $(d_r - 1)$ other points to determine the polynomial $q_r(x)$. Then, for any other node u (including leafs), set $q_u(0) = q_{\text{parent}(u)}(\text{index(u)})$ and choose $(d_u - 1)$ other points for q_u, similar to above.

Whenever u is a leaf node, use $q_u(x)$ to define a secret share $D_u = \frac{q_u(0)}{s_i}$; where $i = \text{attr}(u)$, s_i the randomly chosen secret number from *Setup* and s_i^{-1} the inverse of s_i in $\mathbb{Z}_q^*$.

Return the generated key as $D = \{D_u | u \text{ leaf node of } \Gamma\}$.

Encrypt(m, ω, Params) [2]. Input: Message m, set of attributes ω and public parameters *Params*.

Randomly choose $k \in \mathbb{Z}_q^*$ and compute $C' = k \cdot PK = (k_x, k_y)$. If $C' = \mathcal{O}$, repeat until $C' \neq \mathcal{O}$. k_x is the encryption key and k_y is the integrity key.

Then compute $C_i = k \cdot P_i$ for all attributes $i \in \omega$.

Encrypt the actual message as $c = \text{Enc}(m, k_x)$, generate a Message Authentication Code $\text{mac}_m = \text{HMAC}(m, k_y)$.

Return the ciphertext $CM = (\omega, c, \text{mac}_m, \{C_i | i \in \omega\})$

Decrypt(CM, D, Params) [2]. Input: Ciphertext CM, decryption key D and public parameters *Params*.

Decryption is split into two phases: Reconstructing the curve point C' to get the encryption and integrity keys, and actual decryption of the ciphertext.

For the first phase, define a recursive decryption procedure for a node u DecryptNode(CM, D, u). For leaf nodes with $i = \text{attr}(u)$:

$$\text{DecryptNode}(CM, D, u) = \begin{cases} D_u \cdot C_i \overset{(*)}{=} q_u(0) \cdot k \cdot G & i \in \omega \\ \perp & i \notin \omega \end{cases}$$

Where the equality $(*)$ holds because s_i and s_i^{-1} cancel out:

$$D_u \cdot C_i = q_u(0) \cdot s_i^{-1} \cdot k \cdot P_i = q_u(0) \cdot s_i^{-1} \cdot k \cdot s_i \cdot G = q_u(0) \cdot k \cdot G$$

For an internal node u, call DecryptNode(CM, D, v) for each of its childen v. If for less than d_u of the child nodes DecryptNode$(CM, D, v) \neq \perp$, return DecryptNode$(CM, D,) = \perp$. Then let ω_u be an arbitrary subset of child nodes of u, where for all $v \in \omega_u$, DecryptNode$(CM, D, v) \neq \perp$. Then DecryptNode(CM, D, u) is defined as follows, where $i = \text{index}(v)$, $\omega'_u = \{\text{index}(v) | v \in \omega_u\}$.

$$\begin{aligned} &\text{DecryptNode}(CM, D, u) \\ &= \sum_{v \in \omega_u} l_{\omega'_u, i}(0) \cdot \text{DecryptNode}(CM, D, v) \\ &= \sum_{v \in \omega_u} l_{\omega'_u, i}(0) \cdot q_v(0) \cdot k \cdot G \\ &= \sum_{v \in \omega_u} l_{\omega'_u, i}(0) \cdot q_{\text{parent}(v)}(\text{index}(v)) \cdot k \cdot G \\ &= \sum_{v \in \omega_u} l_{\omega'_u, i}(0) \cdot q_u(i) \cdot k \cdot G \\ &\overset{(*)}{=} q_u(0) \cdot k \cdot G \end{aligned}$$

The equality $(*)$ holds because $\sum_{v \in \omega_u} l_{\omega'_u, i}(0) \cdot q_u(i) = q_u(0)$ is exactly the lagrange interpolation polynomial $q_u(x)$ at $x = 0$ with respect to the points $\{(index(v), q_v(0)) | v \in \omega_u\}$.

This means for the root r of the access tree Γ, we have

$$\text{DecryptNode}(CM, D, r) = q_r(0) \cdot k \cdot G = s \cdot k \cdot G = (k'_x, k'_y)$$

With k'_x the decryption key for m and k'_y the integrity key. Therefore now decrypt $m' = \text{Dec}(c, k'_x)$.

Now check if $\text{HMAC}(m', k'_y) = \text{mac}_m$. If yes, the ciphertext has been correctly decrypted and was not tampered with. Return m', otherwise return $\perp$.

3 Challenges encountered during the implementation

3.1 Rust specialties

The borrow checkaaa...

3.2 Lack of Operating System

This means lack of

- allocator and standard library (replaced by `core`, but much less powerful)
  - any dynamically allocated data structures
  - No Vectors and HashMaps Vectors and HashMaps
  - no easy implementation of (access-) trees using recursive enums (as these would become infinitely large when not using indirections)
  - most dependencies depend on `std` in some way, so a lot more work to make all those independent
  - Solution: `heapless` crate and linear representation of access trees (TODO make bounds on size of data types configurable)
- Random Number Generation
  - Problem: need cryptographically secure randomness, but have no OS randomness pool
  - can't just use `/dev/urandom` to get randomness (there is no such thing as a file anyway)
  - standard implementations of `random` resp. `getrandom` crates don't work (rely on stdlib)
  - need own randomness source
  - nrf50 series provides hardware RNG, but we need Rust to be able to interface with this

  - probably need own implementation of `getrandom` crate for Zephyr OS or bare-metal nrf50

- Unit testing abilities

3.3 Performance Limitations

That is, CPU speed (64MHz) and RAM size (..KiB?).

- probably too slow for computing actual pairings
- Solution 1: Scheme without pairings (Yao et al 2015)
- Solution 2: Still do pairings, but hyper-optimize (see TinyPBC on AVR)

List of Figures

List of Tables

Bibliography

[1] V. Goyal, O. Pandey, A. Sahai, and B. Waters, “Attribute-based encryption for fine-grained access control of encrypted data,” in *Proceedings of the 13th ACM conference on Computer and communications security - CCS '06*, Alexandria, Virginia, USA: ACM Press, 2006, pp. 89–98, ISBN: 978-1-59593-518-2. DOI: 10.1145/1180405.1180418.

[2] X. Yao, Z. Chen, and Y. Tian, “A lightweight attribute-based encryption scheme for the internet of things,” *Future Generation Computer Systems*, vol. 49, pp. 104–112, Aug. 1, 2015, ISSN: 0167-739X. DOI: 10.1016/j.future.2014.10.010.

[3] A. Shamir, “How to share a secret,” *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979, ISSN: 0001-0782, 1557-7317. DOI: 10.1145/359168.359176.

[4] J. Katz and Y. Lindell, *Introduction to modern cryptography*, second edition, ser. Chapman Hall, CRC cryptography and network security. Boca Raton ; London ; New York: CRC Press, 2015, xx, 583 SeitenIllustrationen, ISBN: 978-1-4665-7026-9 1-4665-7026-1.

[5] L. C. Washington, *Elliptic curves: number theory and cryptography*, 2nd ed, ser. Discrete mathematics and its applications. Boca Raton, FL: Chapman & Hall/CRC, 2008, 513 pp., OCLC: ocn192045762, ISBN: 978-1-4200-7146-7.

[6] A. Menezes, “An introduction to pairing-based cryptography,” in *Contemporary Mathematics*, I. Luengo, Ed., vol. 477, Providence, Rhode Island: American Mathematical Society, 2009, pp. 47–65, ISBN: 978-0-8218-3984-3 978-0-8218-8156-9. DOI: 10.1090/conm/477/09303.

[7] I. Blake, G. Seroussi, and N. Smart, Eds., *Advances in elliptic curve cryptography*, Lecture note series. Cambridge [u.a.]: Cambridge University Press, 2005, XVI, 281 S. ISBN: 0-521-60415-X.

[8] S.-Y. Tan, K.-W. Yeow, and S. O. Hwang, “Enhancement of a lightweight attribute-based encryption scheme for the internet of things,” *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6384–6395, Aug. 2019, ISSN: 2327-4662, 2372-2541. DOI: 10.1109/JIOT.2019.2900631.