Experimental Performance Evaluation of Pairing Based Cryptography on Constrained Devices

Olthuis, Jorrit J.

Award date:
2022

Experimental Performance Evaluation of Pairing Based Cryptography on Constrained Devices

J.J. Olthuis

June 28, 2022

Abstract

Pairing-based cryptography has allowed for many new types of cryptographic protocols. It is a cryptography operation on elliptic curves that maps every two points from two groups to a third point in another group. This type of primitive allows for wide ranges of new or much improved algorithms. The challenge with pairing-based cryptography is the complexity of the required computations, leading to high execution times.

In principle, such a requirement makes pairing-based cryptography poorly suited to be executed on constrained devices, those without much computational power, such as devices used in IoT. Furthermore, these devices commonly have other limitations, such as a fixed amount of energy or time available to run.

It sounds like these properties make it impossible to run pairing-based cryptography on constrained devices. However, this is not necessarily the case.

This report analyses the resource requirements of pairing-based cryptography on constrained devices. Specifically, the execution time and energy usage of the computation is considered, as well as the communication of the operands using common IoT protocols such as IEEE 802.15.4. Two different types of elliptic curves at different levels of security are evaluated.
The results show the effect of hardware acceleration on the resource usage, and which part of the resources is taken up by communicating the inputs or results. The conclusions can be used by developers to determine if applying pairing-based cryptography on constrained devices fits within their use case and constraints.

Contents

1 Introduction **5**

2 Background **7**
2.1 Pairing-based Cryptography . . . 7
2.1.1 (Bilinear) Pairing . . . 7
2.1.2 Applications . . . 8
2.1.3 Security level . . . 8
2.2 Hardware Platform . . . 9

3 Related work **11**

4 Deployment **13**
4.1 Hardware . . . 13
4.1.1 Requirements . . . 13
4.1.2 TI CC2538 . . . 13
4.1.3 Zolertia RE-Mote . . . 14
4.2 Software . . . 14
4.2.1 Contiki-NG . . . 14
4.2.2 PBC . . . 15

5 Results **17**
5.1 Measurements . . . 17
5.1.1 Time measurements . . . 17
5.1.2 Energy measurements . . . 18
5.2 Hardware acceleration . . . 20
5.2.1 Configuration 1 . . . 23
5.2.2 Configuration 2 . . . 24
5.2.3 Configuration 3 . . . 25
5.2.4 Configuration 4 . . . 25
5.2.5 Discussion . . . 26
5.3 Energy measurements . . . 27
5.3.1 Barreto-Naehrig curves . . . 27
5.3.2 Supersingular curves . . . 27
5.4 Communications . . . 27
5.4.1 Test setup . . . 29
5.4.2 Timing . . . 30
5.4.3 Energy . . . 30
5.4.4 Discussion . . . 33

6 Conclusions **35**

7 Future Work **37**

A Curve parameters **45**
A.1 BN 53 . . . 45
A.2 SS 318 . . . 45
A.3 SS 512 . . . 45
A.4 SS 700 . . . 46

B Detailed callgrind graphs **47**

1 Introduction

Pairing-based cryptography (PBC) is an area of cryptography that makes use of pairing operations on elliptic curves. A pairing is an operation that is possible on certain elliptic curves, where each pair of two elements from two different groups maps to an element of a third group [25].

Initially, pairings were used to enhance attacks on cryptographic primitives like the discrete logarithms [11]. Since, various applications have been proposed to build new protocols based on pairings. Some provide a more efficient version of existing features [8], however there are also new protocols which were previously impossible without pairings [5, 6, 9].

The software processing required to achieve pairings is generally considered to be computationally intensive [18]. Additionally, one has to take into account the energy and time required to communicate these pairings using the limited available bandwidth. The current consensus is that pairing operations are in general too expensive to execute on constrained devices [44].

This project aims to provide a comprehensive performance assessment of pairing operations on constrained devices. Its goal is to provide baseline from several perspectives on the feasibility of pairings on IoT devices. Using this baseline, one can estimate how feasible using pairing-based cryptography based algorithms are, given the requirements of their project.

In order to achieve this comprehensiveness, this report evaluates many different configurations: different curves at various security levels are used, and multiple types of hardware acceleration are tested. There is also a wide selection of tests, such as the time and energy to perform a pairing, or the initialisation. Also, the required bandwidth for communication is evaluated. None of the existing literature combines time and power measurements for both a pairing operation and the corresponding communication. Also the incorporation of this type of hardware acceleration is unique.

The remainder of this report is structured as follows. Section 2 provides some background information on pairing-based cryptography and the constrained hardware used in this project. Next, section 4 provides more detailed information on the hardware and software setup used for testing. Similar research has been done before, these are compared in section 3. This is followed by the results of our testing in section 5. Finally, this report is concluded in section 6 and section 7 makes some recommendations for the future.

2 Background

This sections provides some background information for this project. It covers both pairing-based cryptography and its applications, and the constrained device on which it should run.

2.1 Pairing-based Cryptography

The first uses of bilinear pairings in security were the MOV [27] and FR [14] attacks, using the Weil and Tate pairings, respectively. Since then, pairings have been used constructively for various different applications. The first was a three-party one-round Diffie-Hellman key agreement by Joux [21] in 2000.

Security proofs of protocols based on pairings are all based on variants of the Diffie-Hellman problem [11] (i.e., the discrete log problem). Depending on the situation, it may be using the decisional[1] (DDH) or computational[2] Diffie-Hellman (CDH) problem.

2.1.1 (Bilinear) Pairing

In general pairings are defined as follows

$$e : G_1 \times G_2 \longrightarrow G_T$$

In some cases the two input elements are part of the same group, in which case the pairing is symmetric.

$$\hat{e} : G \times G \longrightarrow G_T$$

in both cases the inputs need to be elements in additive groups of prime order. G_T must be a multiplicative group of prime order. There are three additional conditions required [26] for e or $\hat{e}$ to be a bilinear pairing that we can use for security algorithms:

- bilinearity: $\forall_{R,S,T \in G} : e(R+S,T) = e(R,T)e(S,T)$ and $\forall_{R,S,T \in G} : e(R,S+T) = e(R,S)e(R,T)$
- non-degeneracy: $\forall_{P,Q \in G} : e(P,Q) \neq 1$
- e can be computed efficiently

These conditions lead to several other mathematical properties, as described by Menezes [26]. For more information on pairing in cryptography, the reader is referred to Meffert [25].

Even though (bilinear) pairings can be performed on groups in general, they are generally used in combination with an elliptic curve over finite fields[3]. In general elliptic curves follow the formula $y^2 + cxy + dy = x^3 + ex^2 + ax + b$, however we only look at curves of the form $y^2 = x^3 + b$, and supersingular curves of the form[4] $y^2 = x^3 + ax + b$.

[1] Whether the attacker has a non-zero "advantage" over randomly guessing to distinguish between two ciphertexts.

[2] An attacker cannot perform a polynomial-time computation to break the Diffie-Hellman key agreement protocol.

[3] A finite field is a finite set of elements on which the operations multiplication, division, addition, and subtraction are defined. The order is given by the number of elements in the field, which is always a prime power ($q = p^k$).

[4] Only curves with characteristic not equal to 2 or 3 can be written in this form.

Supersingular pairings Pairings on supersingular curves are a subset of pairings. Elliptic curves of the form $E/\mathbb{F}_q$ are supersingular when, as described by Freeman et al. [13], q and t are *not* co-prime, with $t = \#E(\mathbb{F}_q) - q - 1$.
The supersingular curves used in this report are of the form $y^2 = x^3 + x$ with embedding degree $k = 2$. The pairings on this curve are symmetric.

Barreto-Naehrig pairings Pairings on curves introduced by Barreto and Naehrig [4] are commonly referred to as BN pairings. T In their work, they introduced a way to easily generate curves with embedding degree $k = 12$. Higher embedding degrees offer more security, but are generally more complex to compute. BN curves are of the form $y^2 = x^3 + b$ [24] and are similar to Miyaji-Nakabayashi-Takano (MNT) curves [13].

2.1.2 Applications

Identity-based encryption (IBE) is a concept where the decryption key is given by one's identity (such as their email address). The Boneh-Franklin [6] scheme uses Weil pairings and was, at that time, the only known way to fully implement IBE.

A more complex version of IBE is attribute-based encryption (ABE) [5]. In Ciphertext Policy-ABE, each user has a set of "attributes", based on their person (e.g., job function, experience, department). Using these attributes, the encrypter of a message can specify which attributes are required to decrypt the message using a combination of binary AND and OR operators.

Another application of pairings are various types of signatures. Some work has been able to develop schemes that generate shorter signatures [8] than existing schemes, whereas there are also new applications of signatures like group signatures [7].

A more extensive overview of different applications using PBC is given by Dutta et al. [11]. All the applications above rely on some form of pairing computations. In order to execute these pairings on constrained devices, section 2.2 will evaluate the constraints and requirements.

2.1.3 Security level

A security level is a number, generally expressed in bits, that indicates the theoretical amount of security a cryptographic algorithm offers. Within the same class of algorithms, this allows one to compare security to, for example, parameter size or execution time.

There are various definitions of security levels, which makes comparisons not always trivial. Symmetric and asymmetric cryptography characterise the theoretical security provided by a cryptographic primitive using the security level λ to indicate approximately 2^λ computations required to break the primitive [23].

Table 1: Table of examples of pairing friendly elliptic curves including their parameters, size and security level. A more extensive overview is available in the IETF "Pairing-Friendly Curves" draft [33].

Name		**Size** q	**Symmetric security level**	$k \log(q)$ **security level**
ISO [20]	BN 384	384	128	4608
ISO [20]	MNT 256	256	128	1536
ISO [20]	Freeman 256	252	128	2520

More relevant to PBC is the elliptic curve definition of security level: *"the bit size of the largest prime subgroup"* [30]. Several publications also estimate a relationship between elliptic curve security levels and asymmetric, specifically RSA, security levels [15, 30]. For pairings, this means the security level is $k \log_2(q)$, with embedding degree k and elliptic curve field size q. The embedding degree causes elements to be mapped onto a larger range, thereby increasing security [24].

Note that the security level only gives an indication of how much security might be obtained using a particular method. For example, as mentioned by Lynn [24], the field size protects against index calculus attacks (and influences the security level) and the elliptic curve group size protects against Pollard rho and lambda attacks (but does not influence the security level). Ideally, one should use standardised and verified parameters to ensure proper security. Table 1 shows some standardised curves that are well suited for pairing operations.

2.2 Hardware Platform

This project is evaluating pairing-based cryptography on constrained devices. IoT devices are always constrained in some form [38]. We use the concept of IoT device and constrained device interchangeably.

There are a few constraining factors taken into account in this project. The next paragraphs each discuss one of the constraints to be considered for IoT devices.

Energy constraints Many IoT applications, for example in wireless sensor networks (WSNs), the devices are primarily energy constrained, since they commonly run on battery power. Energy usage is the main criterion to determine the success of running PBC operations on a device. The components drawing the most power are the transceiver and CPU. Hence, the focus should be on minimising energy usage of these components.

Energy usage is the power draw of a device over time. The power draw of a the CPU may not always be easy to affect, aside from going into sleep mode, therefore this project regularly focuses on minimising execution time instead.

IoT devices almost always require some form of communication. Due to their limited nature, they generally depend on a third party to get instructions, or to process and store their measurements. For the results to be as generically applicable as possible, we evaluate energy usage when communicating raw input and output of pairing operations in section 5.4.

Computational constraints One of the consequences of limited energy usage is a limited processor. Many IoT devices use a processor based on the ARM Cortex M processor family [45]. ARM licenses their processor designs to be used in products by many different manufacturers, and are characterised by their 32-bit Reduced Instruction Set Computer (RISC) architecture and low power usage.

The best way to reduce power draw of a processor, is to reduce the number of features it has, thereby having to power fewer transistors at once. The consequence of this is that RISC supports much fewer instructions than traditional Complex Instruction Set Computer (CISC) PC processors. It generally does not offer hardware acceleration (e.g., for media encoding/decoding), and it has much less memory to work with. The last of which is a constrained discussed later on.

The inherent problem with limited computation power is the search for a balance to reduce energy usage. A low-power processor may take a very long time to perform a computation thereby, using a significant amount of energy. A faster computation can be achieved using a more power-hungry processor, where the time reduction may or may not compensate for the higher power draw. In most cases a large part of this decision has already been made by the CPU designer, only leaving settings like clock speed and sleep modes for the user to decide [41].

Storage constraints Apart from a constrained CPU, many constrained devices are also limited in their RAM and ROM. Modern IoT devices commonly only have very limited RAM available to store variables and the program state [39], commonly only several kilobytes. ROM is used to store the program description and constant numbers, and is generally available in greater quantities than RAM, albeit still constrained [39]. Many devices have less than a megabyte of ROM.

Practically, these storage constraints mean that no conventional operating systems can run. In turn, this creates challenges to run complex programs. Section 4.2 discusses how this problem is resolved such that it still possible to execute pairing operations.

Communication bandwidth constraints There are multiple factors to the limited communication bandwidth available. The power draw of the transceiver: which is one of the components with the greatest energy usage [12]. Using less bandwidth means the transceiver is active for shorter periods of time.

Another factor is the communication protocols used in IoT are constrained in the number of bits per second they can transmit. The commonly used IEEE 802.15.4 [19] only has a theoretical maximum bitrate of 250 kbps, and even less if signal loss, data integrity or protection, and an application layer protocol.

Time constraints There are many applications in IoT where there is a real-time component [2]. This limitation is closely connected to the computational constraints. In some cases even, one may consider to trade less computation time for more energy usage, for example by increasing the processor frequency.

In order to objectively compare pairing operations, one needs to be able to measure the execution time with sufficient precision and accuracy. There is some variance to be expected from the device itself, the random input, and the measurement setup. We aim to achieve an overall accuracy better than 1% variation within one configuration. Section 5.1.1 describes the setup to measure execution time of operations.

Cost constraints Finally, the cost of IoT devices is an important factor. In most cases they are either deployed in large numbers (such as in a wireless sensor network), or are a small component inside a larger product. In both cases, keeping cost low is vital. Luckily, less advanced processors are generally not only more power efficient, but also cheaper. Devices are regularly tailored to specific use cases, including or leaving out functions like buttons, LEDs, communication frequencies, etc. based on the specific application.

Table 2: Comparison between various related works. A ✓ indicates that the work covers a topic. A ✗ indicates that the work does not satisfy the property.

	Unbiased	Time measurements	Energy measurements	Bandwidth considerations	Hardware tested	Hardware acceleration
This work		✓	✓	✓	✓	✓
Yu and Li [46]	✗	✓			✓	
Zhou and Su [47]	✗	✓			✓	
Sankaran [35]	✗	✓	✓			
Ramachandran et al. [32]		✓			✓	
Ometov et al. [29]		✓	✓		✓	
Canard et al. [10]	✗	✓			✓	
Szczechowiak et al. [40]		✓	✓[5]		✓	
Salman et al. [34]		✓	✓		✓	✓
Hajny et al. [17]		✓			✓	

3 Related work

Measuring performance of pairing operations on constrained devices has been done numerous times in literature. Table 2 shows a summarised comparison of several publications covered in more detail in the remainder of this section.

There have been many publications presenting new applications of PBC for constrained devices. Many of these also present execution time measurements [46, 47] and sometimes power [35] measurements (either on real devices, or in simulation) for their configurations. These measurements are however not nearly comprehensive enough to draw generic conclusions from. Moreover, since they are trying to show the effectiveness of their own method, their measurements may be biased, hence the ✗ in table 2.

There are more comprehensive studies such as by Ranmachandran et al. [32] or Ometov et al. [29]. The latter analyse the time to perform traditional (RSA, AES, SHA) cryptography and compare it to elliptic curve and pairing operations. Ometov et al. do this on 9 different wearable IoT devices, albeit significantly more powerful than evaluated in this report: they have at least 1000 times more RAM, and all use at least the significantly more performant ARM Cortex A series.

Rather than thoroughly evaluating the performance of PBC on constrained devices, Canard et al. [10] look for alterations in the computations to make pairing operations more feasible. They consider to delegate pairing computations to a more powerful peer, but recommend replacing the bilinear pairing operations by different elliptic curve operations. Their pairings take place on the Barreto-Naehrig elliptic curve, but they seem to use a non-standard pairing operation.

[5] Does not measure energy usage, only estimates it.

Szczechowiak et al. [40] provide the most comprehensive evaluation found of PBC on constrained devices in a WSN. They evaluate pairings on both supersingular and MNT curves on three devices with 8, 16, and 32-bits processors, respectively. For each, they have developed an implementation in C code, and one in assembly. Their comparison metric is the number of instructions executed, and their assembly implementation is significantly faster than the C version. Results for energy usage are provided, but come from simulations. Their evaluation unfortunately does not consider communication cost (neither in time nor in energy), even though this an extremely relevant factor for WSNs.

Salman et al. [34] take a different approach, and evaluate performance when introducing hardware accelerated operations: they are using a hardware-based Montgomery multiplier to accelerate pairings on BN curves. Their configuration is not supported by generic processors, as they make use of an field-programmable gate array (FPGA) connected to the main CPU, thereby severely limiting possible applications.

Finally, Hajny et al. [17] evaluate pairings on smartcards and Raspberry Pis. The APIs of many smartcards do not fully support elliptic curves, and none supports pairing operations. A custom pairing implementation, like Szczechowiak et al. [40] is not considered. The Raspberry Pis perform the elliptic curves using five different libraries. They only evaluate timing, and do not consider the power usage of these operations.

This work evaluates the performance of pairing operations on two different curves on a modern 32-bit constrained device, using a commonly used [17, 22, 31] library. Moreover, it determines the possible advantages of hardware acceleration to replace parts of the pairing computations. Additionally, we provide a much more comprehensive overview by including the cost to communicate the inputs and results of these pairing operations in the analysis, unlike any of the other publications.

4 Deployment

This section explores the setup used for the performance evaluation carried out in this project. It motivates the considerations and choices made for hardware and software that are used to obtain the results in section 5.

4.1 Hardware

4.1.1 Requirements

There is a list of requirements to be satisfied by an hardware platform in order to be able to be used in the tests in section 5. These are based on the constraints previously listed in section 2.2.

- **Computation** Given the situation described in section 2.2, the goal is to find a similarly constrained processor. Processors based on the Cortex-M family of CPUs from ARM [45] are the most straightforward candidate. Low power usage of the processor is one of the main criteria.
- **Sleep** The processor must be able to go into sleep mode to further reduce power draw when idling, or waiting for other processes to finish.
- **Hardware acceleration** Hardware built to perform very specific operations using less resources may be built into the processor.This allows us to test if it is possible to perform pairing operations significantly faster, or using less energy when they are (partially) performed in specialised hardware.
- **Development board** This project requires a full development board, rather than only a CPU or a board intended for deployment. While such a board may have many features that will go unused, is more expensive, and may have higher energy usage, it also makes testing much easier.
- **Communication** The board needs to provide some form of communication. Ideally it should support IEEE 802.15.4 [19], but any standardised form of wireless communication commonly used in IoT will suffice.
- **Software support** The software chosen in section 4.2, should be able to run on the selected device. While there may be some flexibility in choosing software based on the device, a well supported and documented device is more convenient in general.

4.1.2 TI CC2538

Texas Instruments has several CPUs that satisfy the requirements above. For example, the CC13x0, CC16x0, and CC2538. All these CPUs are based on an ARM Cortex M3 core [45] that is power efficient and provides some form of wireless communication. There are also cryptographic hardware accelerators, the CC2650[6] for example has an AES module. The CC2538 was chosen for this project because of its generic hardware accelerators and relatively large flash and SRAM. Additionally, the CC2538 has a 2.4 GHz transceiver built-in to communicate using the well known 802.15.4 [19] standard.

[6] `https://www.ti.com/product/CC2650`

4.1.3 Zolertia RE-Mote

One of the manufacturers that uses the CC2538 is Zolertia[7]. They have several development boards, most of which use their *Zoul* module. Besides being based on the CC2538, it contains a CC1200 to gain sub-GHz wireless connectivity. It features two buttons, a three-coloured LED, an SD card slot for storage, various GPIO pins, and can run on a battery. Our tests are performed using the RE-Mote Revision B1, of which the details are published on GitHub[8].

4.2 Software

Several software components are necessary to be able to perform pairing operations on constrained devices. The operating system (OS) provides a basic feature set to build programs upon. Such a set generally includes features like a hardware abstraction layer such that the OS can be used on different boards, a scheduler to manage multiple tasks, and a (wireless) communication library. In our case, we also require support for hardware acceleration for (parts of) pairing computations.
The operating system is generally combined with a compiler to generate executables from the C source code. GNU offers an ARM cross-compiler called `arm-none-eabi-gcc` which can be used for the CC2538.

Besides the OS, we need a software implementation of pairings. C libraries exist to provide this functionality, offering various ECC curves and security levels.

Based on the operating system and libraries, an implementation[9] was constructed which will be used in the tests in section 5.

4.2.1 Contiki-NG

Contiki-NG [28] is the *Next Generation* version of Contiki OS. This new version focuses on the future use of Contiki, and replaces or removes the legacy components of the old Contiki OS. It has much more detailed documentation than the original Contiki OS. Important for our application is the support for hardware accelerated operations on the CC2538. Contiki-NG supports 9 different CPUs, some of which are used in multiple boards.

Cooja is a simulator that comes from the original Contiki OS. Important to note is that this is not an emulator, and behaviour may be different between simulation and execution on the real device. The main purpose of Cooja is to simulate a network of devices, including their communication. It can obtain measurements from these simulations, such as power usage estimates. Unfortunately, Cooja does not by default support the CC2538.

RIOT OS RIOT OS [3] was also considered as a possible operating system. This project started in 2008. Applications can be programmed in C, C++ or Rust. At the time of writing, it has support for 238 boards, using 68 different CPUs. It can emulate using Renode[10] and Qemu[11], which make testing and debugging significantly easier. Renode supports the CC2538 CPU, and allows one to connect to GDB (GNU Debugger).

[7] `https://zolertia.io`
[8] `https://github.com/Zolertia/Resources`
[9] `https://gitlab.com/jorritolthuis/pairing-contiki`
[10] `https://renode.io`
[11] `https://www.qemu.org`

RIOT OS however does not support hardware acceleration for the CC2538, which would require more work to port these features from the Contiki-NG implementation. Additionally, Girgenti et al. [16][12] has already used Contiki-NG to perform pairing operations on the CC2538, which makes that a much more convenient starting point.

4.2.2 PBC

There are many libraries that offer pairing operations, some of which have been compared by Hajny et al. [17]. The main requirement is that the library is written in C or C++, such that it works well with Contiki-NG. Support for various curves is also a plus, as this creates the opportunity for comparisons.

The pairing-based cryptography (PBC) library by Ben Lynn [24] is written in C and provides seven different types of curves. Additionally, it is used by many different projects [17, 22, 31] and has various derivatives in different programming languages[13]. As shown by Hajny et al. [17], PBC is a rather inefficient library, and can hence provide us with worst case measurements.

Of the seven supported curves, section 5 will look at type A ($y^2 = x^3 + x$, also known as supersingular (SS) curves) and type F ($y^2 = x^3 + b$, also known as Barreto-Naehrig (BN) curves). For each of the curve types, PBC can generate its own parameters. How security holds up for these generated parameters is unclear.

PBC was not designed with IoT devices in mind. It relies heavily on dynamic memory allocations and even overallocates memory in places: *"Instead of right shifting every iteration, I allocate more room for the z array."*. The heavy reliance on function pointers makes things worse, since it is significantly more difficult - both for a compiler and a human - to determine what code is actually required and used. Finally, the library does not compile using its built-in build process when modifying the setting for ARM[14].

Warning

The PBC library is not actively maintained. The last release was in 2013 (9 years ago at the time of writing). Since, there have been a few commits on GitHub[a], but no new release. Additionally, no messages on the mailing list[b] have recently been answered.
Given the lack of maintenance, it should not be trusted for real world computations requiring actual security. Since this project is not comparing security, but rather performance, the details of the implementation are less relevant.

[a] `https://github.com/blynn/pbc/commits/master`
[b] `https://groups.google.com/g/pbc-devel`

[12] `https://github.com/wellsaid`
[13]
Python: `https://github.com/debatem1/pypbc`
Perl: `https://metacpan.org/dist/Crypt-PBC`
C++: `https://crysp.uwaterloo.ca/software/PBCWrapper/`
Java: `http://gas.dia.unisa.it/projects/jpbc/index.html`
.NET: `https://github.com/ymcrcat/MASHaBLE`
[14] On the website of PBC, a version for ARM by *KISON research group, UOC* is listed. The URL of this is however broken at the time of writing, so using this is not an option.

GMP Pairing computations are performed on ECC curves where operands of several hundreds of bits or more are common. Most recent IoT devices only offer hardware operations up to 32 bits wide. This means that all computations on the large numbers must be split up into smaller computations such that they can be performed in hardware by the CPU.

The PBC library uses the GMP library for functions on arbitrarily large numbers (only limited by RAM size). It also offers arbitrary precision computations, however that feature goes unused in this project, as all numbers are integers[15].

A significant disadvantage of GMP is that it was clearly not designed for IoT devices. While it is optimised in terms of computational power, there was less attention to the code size and dynamic memory allocations. Code size is relevant due to the very limited amount of flash available on the CC2538 (or similar devices): it only has 512 kB available for all the program code, including the libraries. Luckily, the compiler is rather good at distinguishing what is code might be used, and removing anything else. Dynamic memory allocations are generally frowned upon for embedded devices[16], given that SRAM is also a very precious resource (the CC2538 has 32 kB). Dynamic allocations make it virtually impossible to predict if or when the device will run out.

Note that GMP does not make any security claims. Consider the possibility that it leaks secrets through side channels.

[15] Also important to note is that the CC2538 does not support any floating point arithmetic in hardware, regardless.

[16] In favour of static, but deterministic memory allocation.

5 Results

Given the described configuration of the Zolertia RE-Mote (section 4.1.3), Contiki-NG (section 4.2.1), and the PBC library (section 4.2.2), tests are executed in this section. There are four different curve sizes under test. They are based on supersingular and Barreto-Naehrig curves.

For all curves, the parameters were generated using the PBC library. For supersingular curves one can choose the length of the group and field sizes separately. Barreto-Naehrig curves only allow a single security parameter to be set.
In general, the security level is given by the size of $q^k = k \log_2(q)$ with k the embedding degree [4, 22, 30, 37]. However, as noted by Lynn [24], the size of the field is to prevent index calculus attacks, and the size of the group is to protect against Pohlig-Hellman attacks [37].
Appendix A lists the full parameters used for each of the following configurations.

- **SS 512** A supersingular curve with the minimum recommended 512 bits in size. Because this curve has an embedding degree of 2, it has a security level of 1024. The size of the group order is 160 bits, like all supersingular curves, as recommended [24].
- **BN 53** The size of the Barreto-Naehrig curves is limited by the library and RAM of the embedded device. This limits the size of both the field size and the group order to 53 bits. Due to the embedding degree of 12, it provides 636 bits of theoretical security. Important to note is that the group order is also 53 bits, significantly less than the recommended 160 bits, leaving this curve possibly specifically exposed to Pohlig-Hellman attacks [37].
- **SS 318** To compare the timing and power results of BN 53 to a supersingular curve of roughly the same security level, SS 318 also offers 636 bits of security (although the group order is 160 bits here, as recommended).
- **SS 700** Finally, SS 700 is roughly the maximum size[17] that can be handled using the combination of the PBC library and RAM available. With the same embedding degree of 2 as the other supersingular curves, this provides a security level of 1400 bits.

5.1 Measurements

For the further results sections, we are primarily interested in execution time and energy usage. Sections 5.1.1 and 5.1.2 describe, respectively, the setup to measure these properties.

5.1.1 Time measurements

Most devices offer a low frequency timer that can be probed to obtain the time. For example, the TI CC2538 [41] has a 32 kHz `rtimer` for this purpose. The time difference between two measurement points can be used to compute execution time.

Unfortunately, our testing has shown that the `rtimer` does not provide reliable measurements. Figure 1 should show two horizontal lines for two configurations, but it clearly does not: the black and green should together form one horizontal line, and the blue and red line should together form the other horizontal line. The measurements vary from day to day, but they seem consistent within one day. This could be due to many factors, however we rule out the effect of temperature and humidity in the room as these were monitored during the tests.

[17] At size 768 bits, the pairing is not able to complete.

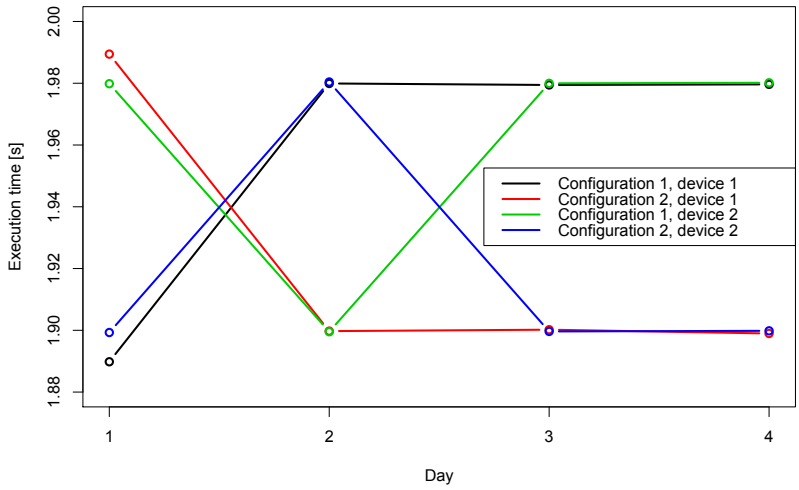

Figure 1: Execution of two software configurations measured over a period of four days. Measured on two instances of the CC2538 using `rtimer`.

The difference between the configurations is the use of hardware acceleration (HWA). The operations in hardware use a different clock. The variance could partially be explained by different variations of the main clock and hardware clock. The CC2538 has RC (Resistor Capacitor) clocks, which can have a lot of variance. However at 32 MHz, we are guaranteed to use the much more stable crystal oscillators, which makes this explanation also unlikely.

An alternative explanation could be the interrupts disturbing the execution. Although one would expect to occur irrespective of the day and sample, rather than clearly separated by day. This hypothesis was tested by disabling the "master interrupt". Unfortunately, the exact same behaviour was observed.

The final solution, motivated by figure 2, is to extract timing measurements from the power measurements. We see that the time obtained using this method are far more constant, within 1%. By inserting sleep periods before and after the pairing operation, we can see and measure the time taken for the execution. The major disadvantage of this method is the resolution, rather than having minimum period of 30 μs with `rtimer`, this setup has a minimum period of 2 ms. The limitations of this setup are discussed in more detail in section 5.1.2.

5.1.2 Energy measurements

In order to measure energy usage, there are two general approaches: Provide the hardware with a known amount of energy stored, and compare this to the remaining energy after execution [1]; or sample the power usage regularly throughout the execution. We opted for the latter due to its relative simplicity. The disadvantage is that this method will miss short peaks, both up and down, in the power draw.

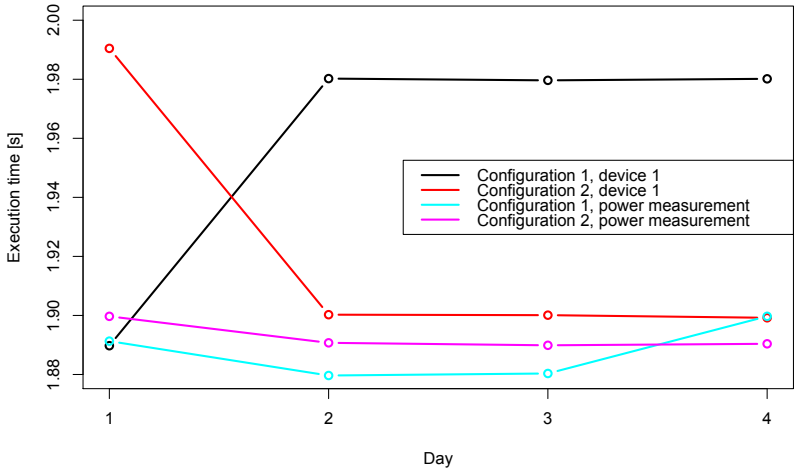

Figure 2: Time measurements compared to time measurements extracted from power measurements over four days on one device.

Figure 3 shows the power measurement setup. It revolves around the TI INA219 Power Measurement IC. This chip measures the voltage drop over a shunt resistor to determine the current going through the IoT device. Additionally, it measures the voltage drop over the device itself. The product of both measurements gives us the power draw.

The INA219 can be configured to work on various ranges of inputs. It can measure at most 32 V and 2 A (i.e., 64 W). Since constrained devices use significantly less than 64 W, we used a configuration with a higher resolution. After configuration, the voltage precision is $\frac{16\ V}{12\ bit} = 4\ mV/bit$, and current precision is $\frac{400\ mA}{12\ bit} \approx 10\ \mu A/bit$. Note that $10\ \mu A/bit$ comes at the cost of overflowing before reaching 400 mA, however the board is not expected to reach these levels anyway.
In total, we get a power draw precision of 0.04 μW at 2 kHz (i.e. the maximum frequency of the ADC) [42]. By default, the INA219 power measurement has a precision 20 times less than the precision of the current measurement, a better resolution can be achieved by multiplying current and voltage manually as a form of post-processing.

The INA219 is connected over an I2C bus to an Arduino Due. AdaFruit provides a library for using the INA219 on Arduino boards [18], which makes them especially convenient. Although any Arduino would work, the Due was chosen because of its high clock speed and large SRAM. To get as close as possible to the maximum frequency of the analog to digital converter (ADC), one needs to buffer the measurements, rather than immediately printing them out to the serial bus.

The program used to perform these power measurements is available on GitLab[19]. It outputs time difference between measurements, and the voltage and current measurements. The measurement accuracy is subject to the accuracy of the shunt resistor, voltage meters, and ADC.

[18] `https://www.arduino.cc/reference/en/libraries/adafruit-ina219/`
[19] `https://gitlab.com/jorritolthuis/pairing-contiki/-/blob/master/getcurrent_csv_arduino.ino`

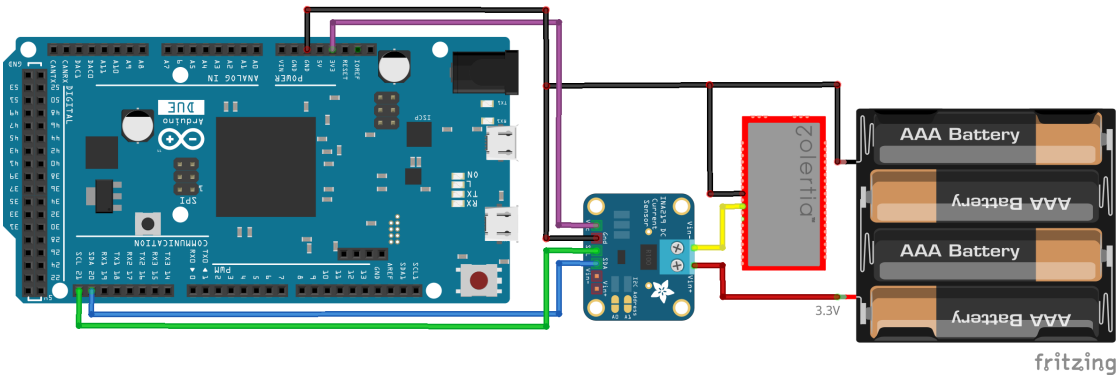

Figure 3: Power measurement setup using the INA219 and Arduino Due. The device under test is the Zolertia board with the red border. The battery pack is illustrative for any DC power supply at 3.3V.

5.2 Hardware acceleration

For each of the previously described curves, a set of tests was executed. The tests consist of an implementation fully in software, and various software implementations augmented by one or more hardware operations. These configurations are compared based on their execution time, which are shown in table 3.

First, we look at figures 4 and 5. These show executions of a supersingular and Barreto-Naehrig pairing, respectively. Figures 6 and 7 show the full execution (including initialisation of the PBC library). Appendix B contains larger and more detailed versions of these figures. Each block represents a function in the code, and has a number indicating the cumulative number of execution steps[20] (inclusive of execution time of sub-functions) of that block. There are arrows between the blocks indicating function calls, and numbers showing the number of calls. The executions were performed on a PC (rather than a constrained device). While there may be some small differences (e.g. due to the presence of certain complex hardware components), the graphs can still be used to identify resource intensive computations.

While the results are primarily based on the execution times without initialisation, it is an important variable to consider. As mentioned before, the PBC library uses a considerable amount of memory. If replacing (or rebuilding) the library is not an option, one may consider to uninitialise the library when not in use, leading to repeated initialisation cycles.

Table 3: Description of four different testing configurations and the section numbers in which the results are presented.

Configuration name	**Description**
Configuration 1 (5.2.1)	Full software
Configuration 2 (5.2.2)	Field element modular inversion acceleration
Configuration 3 (5.2.3)	Field element modular multiplication acceleration
Configuration 4 (5.2.4)	Elliptic curve hardware acceleration

[20] According to the callgrind authors: *"The Ir counts are basically the count of assembly instructions executed"*. [43]

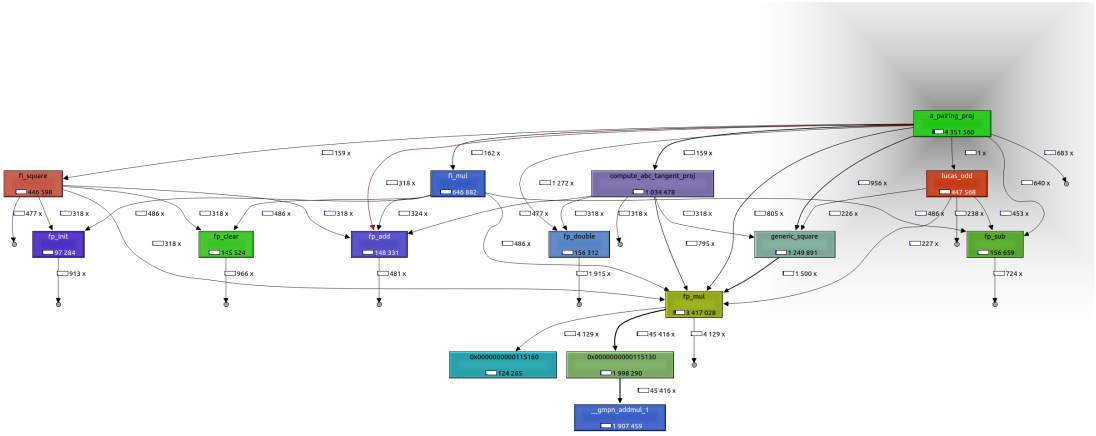

Figure 4: Graph of callgrind [43] execution of a supersingular pairing, indicating which computations take the most time.

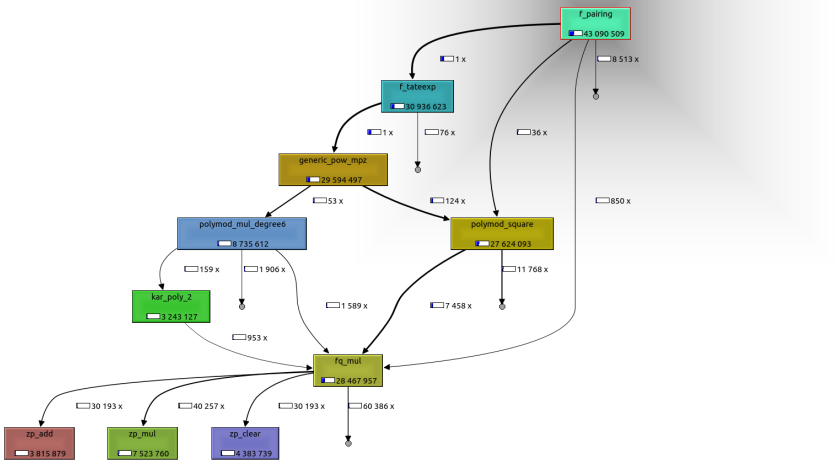

Figure 5: Graph of callgrind [43] execution of a Barreto-Naehrig pairing, indicating which computations take the most time.

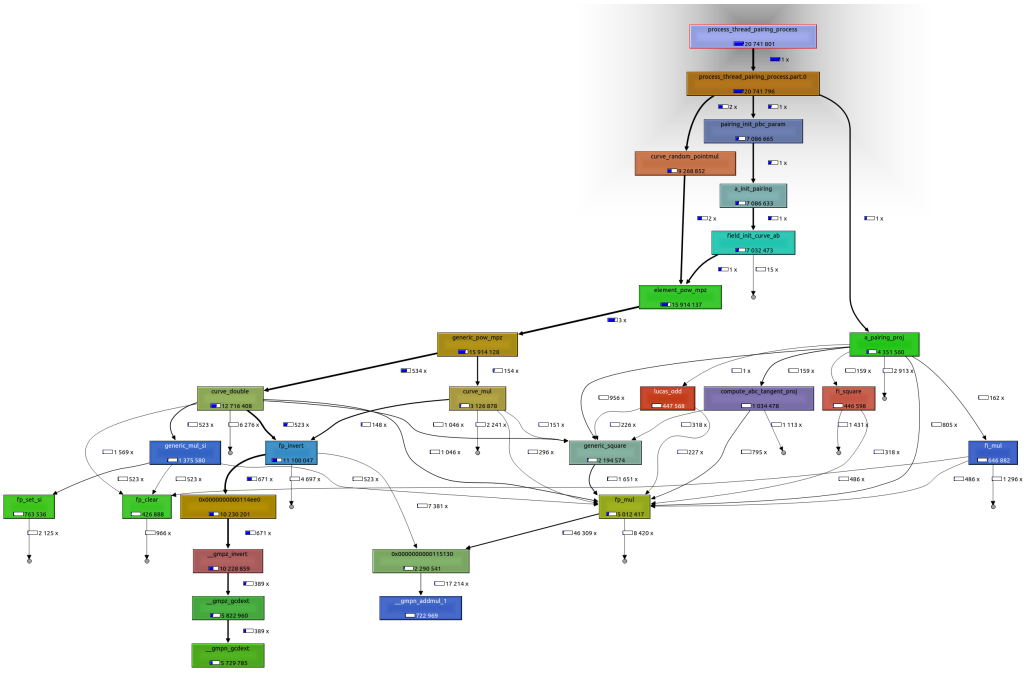

Figure 6: Graph of callgrind [43] execution of a supersingular pairing, including initialisation of the PBC library, indicating which computations take the most time.

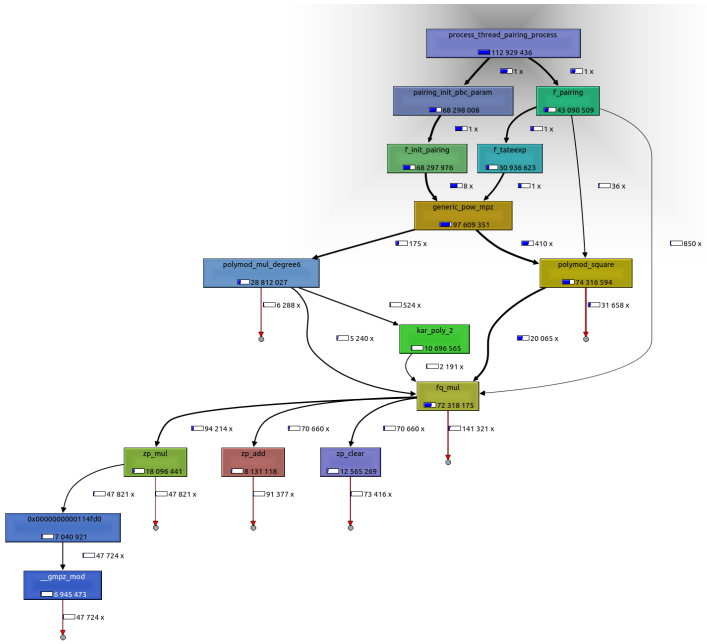

Figure 7: Graph of callgrind [43] execution of a Barreto-Naehrig pairing, including initialisation of the PBC library, indicating which computations take the most time.

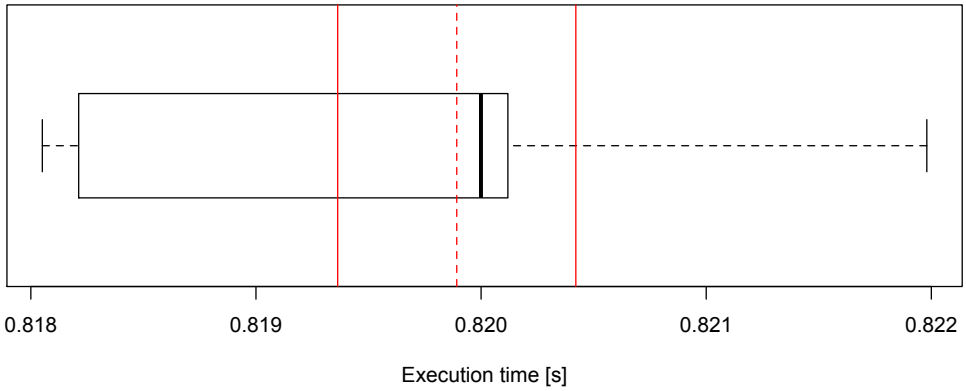

Figure 8: A boxplot indicating the variance of 28 measurements of one configuration, with the mean and 95% confidence interval shown by the red vertical lines.

For each of the timing tests in the following sections a small number of trials was averaged. Figure 8 shows that measurements are very constant, and that therefore a small number of tests is sufficient: the two sides of the 95% confidence interval are only 0.6% away from the mean, showing that we do not need many measurements to be confident in our result: the probability that, when we have two measurements, both are more than 0.06% away from the true mean is 0.06%[21]. All tested combinations of curves and configurations had two or three test samples. The standard deviations were never more than 4 ms, confirming the conclusion drawn from figure 8 and the goal error margin set in section 2.2. The mean values are used in the sections below.

5.2.1 Configuration 1

The reference implementation is a version completely implemented in software. It uses the default pairing functionality in the PBC library, corresponding to the given curve. The input is randomly generated using the random number generator in Contiki-NG. The setup and random number generation time and energy usage are not taken into account, unless explicitly stated otherwise. These times provide the baseline to compare against in the next sections.

This configuration can also be executed on a PC, since it does not rely on CC2538-specific operations. An execution at a security level of 1024 bits is shown in the callgrind figures.

Table 4: Execution time in seconds of one pairing operation in software (i.e. configuration 1).

BN 53	**SS 318**	**SS 512**	**BN 700**
2.581	1.308	1.880	6.714

[21] If the two measurements fall outside different sides of the confidence interval, this error would be noticed by an abnormally high standard deviation.

The execution times are shown in table 4. As expected, larger sizes of the supersingular curves lead to higher execution times. We can also confirm that BN pairings take more time than SS at the same security level, as claimed by the PBC authors.

Additionally, the setup time of both SS and BN curves was measured. Figures 6 and 7 show an execution including both the setup phase and one pairing operation. The setup consists of two parts: initialising the PBC library, as well as generating random input. The setup times are 0.82 seconds and 3.4 seconds for the SS and BN pairings at equal security levels, respectively.

5.2.2 Configuration 2

The first function that we evaluate for hardware acceleration is the modular inversion (i.e. it computes A^{-1} such that $A \cdot A^{-1} \equiv 1 \mod q$). The software implementation uses the extended Euclidean algorithm to perform this computation.

For supersingular pairings, we see in this function under the name `fp_invert` in figure 6. It takes 11.1 million out of 20.7 million samples, leading to a theoretical maximum speed up of more than 50% for the full process, including setup. However, since the function is not shown in figure 4, we should expect limited change for just the pairing operation.

The function is not shown in either call graph of the BN curve, meaning it is not called at all, or does not constitute a significant portion of the execution. No significant change in execution time should be expected.

Table 5 shows the execution times compared to the full software implementation. The relative difference is at most 0.86% slower, which is well within the error margin described in section 5.1.2. This is expected, given `fp_invert`'s absence in figure 4.

In the setup phase of the supersingular pairing, `fp_invert` takes 11.1 million out of 16.4 million cycles, leading to a theoretical maximum improvement of 67%. The setup takes 45% longer than configuration 1: 1.19 seconds. The unexpected time increase could be explained by the fact that modular inversions in software are already very optimised by the extended Euclidean algorithm.

We can conclude from this that there is no advantage *for timing* to using hardware acceleration for modular inversions for pairing operations on both SS and BN curves. There may be an advantage in terms of executable size or memory.

Table 5: Execution time of one pairing operation using hardware acceleration for the `fp_invert` function (configuration 2), compared to a full software implementation (configuration 1).

	BN 53	**SS 318**	**SS 512**	**SS 700**
Configuration 1 [s]	2.581	1.308	1.880	6.714
Configuration 2 [s]	2.562	1.319	1.896	6.747
Configuration 1 [%]	100	100	100	100
Configuration 2 [%]	99.25	100.81	100.86	100.49

5.2.3 Configuration 3

Rather than focusing on the full execution graph, if we instead focus only on the call graph of the pairing function. We see that the `fp_mul` function takes 3.4 million out of 4.4 million cycles for a supersingular pairing in figure 4. This means that there is a theoretical improvement possible of over 75%.

`fp_mul` takes two field elements (from the same field), and multiplies these two together to obtain a third element (also in the same field). The CC2538 does not provide a function to perform this computation directly. Rather, we perform this computations in two steps: a multiplication of two numbers, followed by a modular reduction to ensure the result falls within the field[22].

We verified this acceleration specifically for SS curves by comparing various results between the original software implementation and the new hardware accelerated version.

The results in table 6 show us that replacing one function by a combination of two hardware functions is very ineffective. Rather than reducing execution time, it doubles it. While this does not completely rule out all cases (a very inefficient software implementation or very efficient hardware implementation may have possibilities), it should be taken as a warning not to depend on such a situation before performing tests.

The main conclusion that can be drawn from the data in table 6 is that the software operation to perform modular multiplication is significantly faster than the combination of two hardware accelerated operations. The software implementation is roughly as fast as a single hardware accelerated operation, a similar result to that in section 5.2.2.

The execution time of SS 700 is too large to obtain timing measurements for. The results of SS 318 and SS 512 can be used to make a reasonable estimate for the pattern to extrapolate to SS 700.

5.2.4 Configuration 4

This implementation accelerates various elliptic curve operations. It was developed for the publication by Girgenti et al. [16]. Perazzo et al. [31] used the same acceleration implementation and did not find any advantage for their application (using supersingular curves): *"The BSW scheme does not benefit from hardware acceleration"*. This is expected as none of the accelerated functions (`curve_mul`, `curve_double`, `element_pow_mpz`, and `element_mul_mpz`) are used according to figure 4 in SS pairings.

Table 6: Execution time of one pairing operation using hardware acceleration for the `fp_mul` function (configuration 3), compared to a full software implementation (configuration 1).

	SS 318	**SS 512**
Configuration 1 [s]	1.308	1.880
Configuration 3 [s]	2.783	3.509
Configuration 1 [%]	100	100
Configuration 3 [%]	212.74	186.69

[22]This order of operations is not the most memory efficient: exponentiation by squaring with repeated modulo operations has a lower peak memory usage.

Instead, figure 5 shows that `generic_pow_mpz` is extensively used by BN pairings[23]. This function takes 29.6 million cycles out of the total 43.1 million, and should be able to benefit greatly from this hardware acceleration.

Table 7 confirms our hypothesis: the hardware accelerated version of the BN pairing is 68.4% faster than it was in software only. This is very close to the theoretical maximum of 68.7%, closer than could reasonably be expected. A possible explanation is that this phenomenon is caused by callgrind, as it was executed at a different security level, as well as on a different machine (a PC, rather than a constrained device).

Looking at the setup phase of the SS pairing in figures 6, large parts of the setup are taken up by `generic_pow_mpz`, it takes up to 95% of setup time. Comparing this theoretical maximum with the actual supersingular setup time, we see that the setup phase takes 43% less time than in software. This is a great improvement, given that the theoretical improvement can only be achieved if the HWA takes 0 time for its computation. This shows that in some use cases reinitialising the PBC library can be made feasible by using HWA.

5.2.5 Discussion

While it was shown that hardware acceleration can make pairing operations significantly faster in section 5.2.4, it has to be used under the right circumstances.

Section 5.2.3 has shown that it is not generally worth it to replace a single software operation by a combination of two hardware functions. What exact hardware accelerations are available depends on the CPU. Section 5.2.2 shows some improvement over a software implementation when the replaced function is only used in the setup phase.

In all cases that hardware acceleration is considered, one should also consider whether a large part of the software implementation can be replaced by a single hardware operation. If there is not, one may consider using a different curve, resulting in different pairing operations.

Note that PBC author says the *current implementation* of Barreto-Naehrig pairings is not as fast as supersingular pairings [24]. Whether how much of this difference is due to lack of optimisation and how much is due to inherent computational complexity is unclear. A more optimised version of BN pairings may reduce the relative effectiveness of the hardware acceleration, although it is unlikely to ever be faster than the hardware accelerated version.

Table 7: Execution time of one pairing operation using hardware acceleration for the `element_pow_mpz` and `element_mul_mpz` functions (configuration 4), compared to a full software implementation (configuration 1).

	BN 53	**SS 318**
Configuration 1 [s]	2.581	1.308
Configuration 4 [s]	0.816	1.314
Configuration 1 [%]	100	100
Configuration 4 [%]	31.63	100.43

[23] Due to some problems between different configurations, we only use acceleration for scalar point multiplication: `element_pow_mpz` and `element_mul_mpz` provided by Girgenti et al. [16].

[24] `https://crypto.stanford.edu/pbc/times.html`

For the supersingular curve, further investigation will be performed at its minimum recommended security level of 1024 bits (i.e. SS 512). The fastest implementation was found to be the full software implementation, which will be the configuration used in the next sections.
Unfortunately, the BN curves are not at this minimum recommended level, so it will rather be run at the greatest security level that was supported: 618 bits. As the elliptic curve acceleration showed great results for BN curves, this acceleration will be enabled for the next sections.

5.3 Energy measurements

We evaluate the optimal configurations of SS 512 and BN 53 for energy usage. This is done using the setup described in section 5.1.2.

5.3.1 Barreto-Naehrig curves

As shown in the previous section, BN 53 is the fastest using elliptic curve hardware acceleration. The evaluation in this section is based on this fastest configuration.

Sleep during hardware acceleration While the CC2538 is performing hardware accelerated operations, the main CPU is not performing any operations. This means we can put the main CPU to sleep during hardware acceleration (HWA). The CC2538 user guide [41] describes that it is safe to drop into the power modes PM0 and PM1. PM2 and PM3 cannot be used as those remove power from half of the SRAM and the hardware accelerator module.
Surprisingly, testing shows that this does not significantly improve energy usage. The measured energy savings by going to sleep are less than 1%. This is less than error margin. This may for example be caused by the low frequency of the power measurement setup, or the powered required to switch power modes is offsetting the advantage.

One BN pairing operation on the RE-Mote takes on average 0.127 Joule. This average is based on 10 measurements from two different devices. There is noticeable difference between the two instances of the device, albeit well within the margin of error. Figure 9 shows the spread of the measurements, as well as the difference between the two devices. Note that the full range of the graph is less than 1% of the total energy usage.

5.3.2 Supersingular curves

Section 5.2 did not find a good acceleration for supersingular pairings. Therefore, this section will present the results of the software implementation of the SS 512 pairing.

Like the power measurements of the Barreto-Naehrig pairing, we take the average of 10 measurements. The implementation is software on the RE-Mote board uses on average 0.293 Joule to perform one pairing operation.

Figure 10 shows the distribution of these energy measurements. The 10 measurements consist of measurements taken over several days. Although the relative range of the datapoints is larger, at less than 3%, it still provides a precise measurement of the energy usage for one pairing operation.

5.4 Communications

While the computation is a large part of the energy usage, close to all IoT devices doing pairing operations will have to do some form of communication.

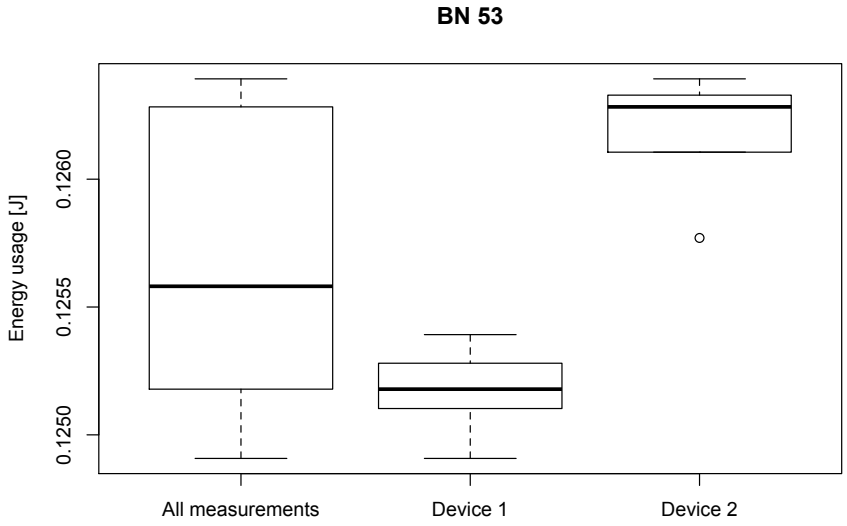

Figure 9: Boxplot to illustrate how the energy measurements of BN 53 with HWA are spread out.

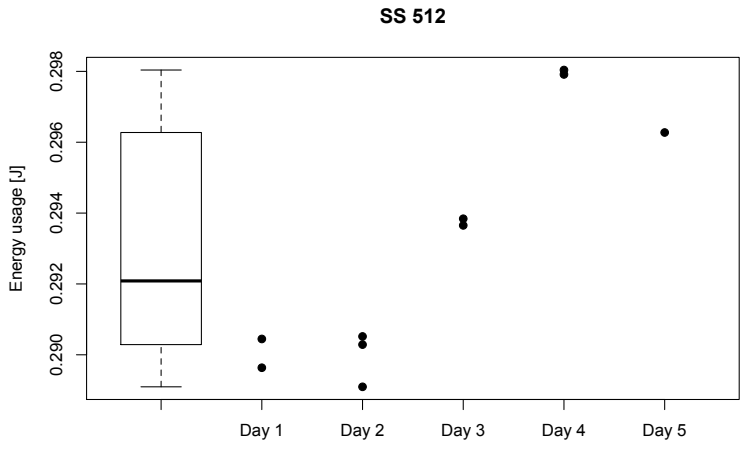

Figure 10: Energy usage of a single SS pairing operation, compared between several days

The exact details of the communication will depend on the application of the pairing operations. Therefore, this section evaluates the time and energy required to generically send and receive inputs and outputs for the pairing operations.

5.4.1 Test setup

Many factors of communication influence time and energy: frequency band, MAC and routing protocol, application protocol, number and location of nodes in the network, interference, etc.

As mentioned before, Contiki-NG offers the Cooja simulator which can simulate networks of devices. This simulation is for example used by Szczechowiak et al. [40]. Besides the fact that Cooja does not support the CC2538 natively, real measurements can be significantly more insightful than simulations.

Contiki-NG offers examples using IEEE 802.15.4 [19] and the constrained application protocol (CoAP) application layer protocol. Our setup consists of two nodes: a client and a server, and they can communicate directly. The client performs a GET request for a certain resource from the server, after which the server responds in one or more packets. The code for these tests is available on Gitlab[25].

IEEE 802.15.4 uses the 2.4 GHz frequency band, just like WiFi and Bluetooth. Most likely, there has been some amount of interference. This interference has not been measured, nor controlled.

Measurements of time and power were both performed using the setup described in section 5.1.2. Going to sleep, like for previous tests caused the device to issue a hard reset. Instead, the power draw of the LED is used to signal the start and end of the communication.

Some transmissions are too fast to be registered with the testing setup, this could be improved as part of future work[26]. Instead, we perform a set of 50 consecutive identical transmissions and average over these. This also helps to compensate for interference. The main problem with this setup is that we measure too much time on the server side: the time during which the client is crafting a new packet, the server is idle. However, this idle time is now also measured. Because of this, server side measurements are excluded from the results[27].

Configurations Measurements have been performed for various configurations. The length used for supersingular curves is 512 bits, for Barreto-Naehrig curves the length is 53 bits. Each curve has specific lengths for the inputs and output, that can be sent and received. Table 8 summarises the different configurations and the corresponding length of the transmitted message in our tests.

Messages are transmitted in hexadecimal text. Each packet can contain at most 64 bytes of payload, which means that the messages in table 8 take 1 or 3 packets to transmit over the CoAP protocol.

[25] `https://gitlab.com/jorritolthuis/comm-contiki`

[26] The expected duration of a one packet transmission, excluding creating or decoding the packet, is 3 ms [36]. The measurement setup has a resolution of 2 ms.

[27] A possible partial solution would be to use a POST request, rather than GET request. The request will contain the data, sent by the client. Alternatively, one can use a faster measurement setup such that single transmission can be measured. Both are left for future work.

Table 8: Message length in bytes, encoded as hexadecimal text.

	SS 512	BN 53
Input 1	192	28
Input 2	192	56
Output	191	168

Table 9: Average time in milliseconds to exchange one message.

	Receive	
	SS 512	BN 53
Input 1	56.6	52.5
Input 2	59.6	57.3
Output	62.8	54.8

5.4.2 Timing

Table 9 shows the execution times to receive the messages as specified in table 8. Compared to the pairing execution times obtained in section 5.2, these times are between 3 and 7% of a pairing operation. When one would have to transmit both inputs and the output, transmissions take a considerable amount of time: up to 17% of the total operation. Important to note is also the lack of a network in this case. In applications such as WSNs communications paths are likely indirect, causing longer waiting periods for both parties.

Figure 11 shows how the message size affects the transmission time. While the smallest transmission size seems slightly shorter than the larger messages, the mean values in the table show minimal difference. This seems to indicate that constant factors play a relatively important role in execution time, compared to operations performed per packet. For example time to create, transmit and decode the GET request packet.

The times are so much higher than the time expected to be taken up by the transceiver, showing that the resource usage of communication cannot simply be ignored. One could measure time, using a much faster setup, to confirm exactly which operation or network stack layer takes the most amount of time.

5.4.3 Energy

For energy usage we look at the same configurations as in section 5.4.2. Table 10 shows the average energy usage in millijoule to request and receive a certain message. Compared to the energy usage in section 5.3, the communication draws less power than the computation. We see a smaller part of the energy usage coming from the communication: between 2 and 5% of the computation energy. The same holds when communicating all the inputs and the output, which would take between 6 and 13% of the total energy.

Interesting to note is that the shortest message is using less energy than the larger messages, 5.7 J compared to 6.5 J. However, the Joule/byte metric is much worse: the long messages average 0.03 J/byte, whereas the short message uses 0.2 J/byte. More research is required to what causes this.

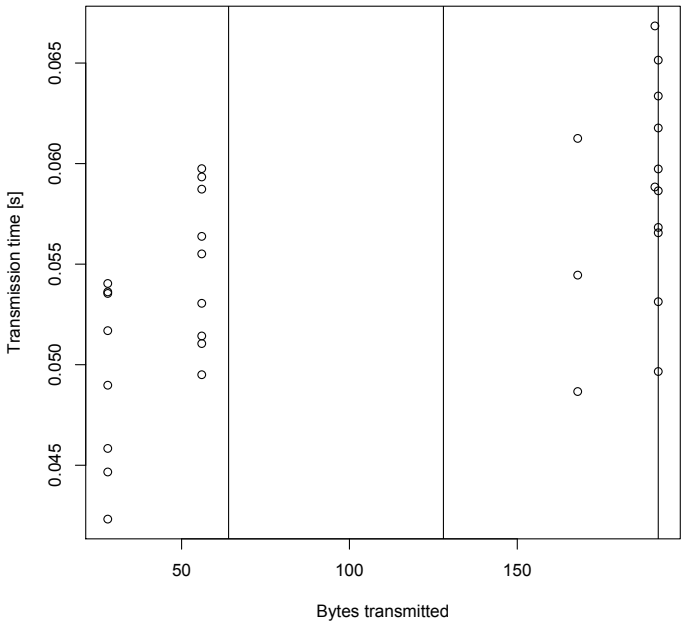

Figure 11: Relationship between transmission time and transmission size, with the vertical lines indicating packet size.

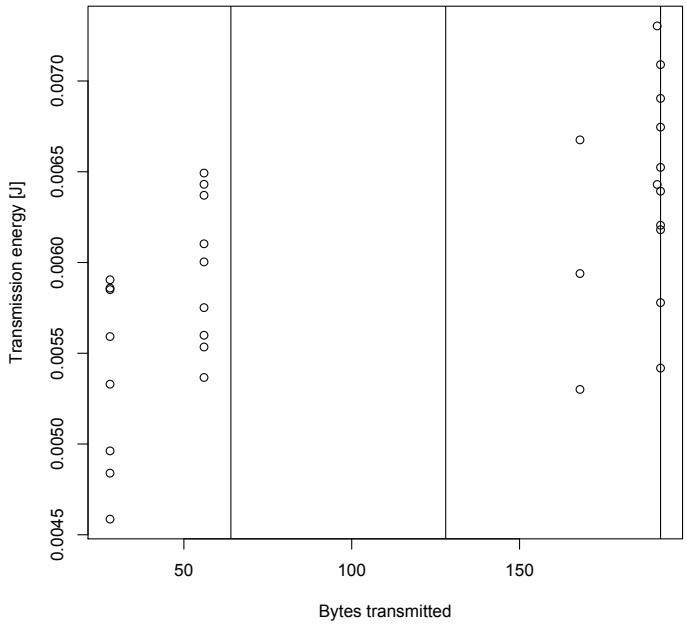

Figure 12: Relationship between transmission energy and transmission size, which the vertical lines indicating packet size.

Table 10: Energy usage in millijoule to exchange one message.

	Receive	
	SS 512	BN 53
Input 1	6.17	5.74
Input 2	6.51	6.22
Output	6.87	5.97

Figure 12 compares the energy usage to the message size. The result is very similar to that of figure 11 because average power draw is within 1% for all measurements. One would expect more variance due to for example interference, however any possible variance is averaged out over 50 transmissions.

5.4.4 Discussion

Due to the large number of different network configurations, communication time and energy may vary wildly. Also the choice including or excluding energy and time to relay message will significantly change the measurements. Regardless, the results shown in sections 5.4.2 and 5.4.3 provide a starting point to analyse the communication.

To start, the observation that the communication of one element takes less than 10% of the computational resources (i.e., time or energy) should be made. If the computation is more optimised, like for the BN pairing, the communication will relatively take more resources. The computation is much easier to optimise than the communication part.

If one is sending or receiving more than one element (e.g. both inputs, or also the output) the relative resource usage of the communication may increase to more than 10%. The exact details are again very much dependent on the optimisation level of the computation.
Another factor can be what exactly is transmitted. Many protocols will likely need more or different information besides the bare pairing operands. This can affect the number of bytes to be transmitted.

What has not been taken into account, but is an important factor, is the possible need for additional encryption. Depending on the application, it may not be safe to send the operands in plaintext over a wireless channel. A simple authentication and symmetric encryption might be sufficient, but does require additional resources.

Finally, a significant variating factor are the protocols, including frequency band, and the network topology. A different frequency will use a different transceiver, which most likely has a different power draw. A star topology has few hops, but likely many medium collisions. A line topology will require interaction with many more nodes, but may suffer less from collisions. These factors are interesting to investigate, but left for future work.

6 Conclusions

This report has investigated the performance of pairing-based cryptography operations on constrained devices. Specifically, the performance of a pairing using the PBC library in Contiki-NG, running on a Zolertia RE-Mote board. The performance of two different curves at a total of four different configurations was evaluated.

Section 5 presented these performance results. The possibility of using hardware acceleration in the Zolertia RE-Mote board was considered to save resources. Apart from a full software implementation, three different accelerations were presented: modular inversion, modular multiplication, and elliptic curve operations.

The results showed little to no execution time improvement for most combinations of curve configuration and hardware acceleration. Only the elliptic curve acceleration on Barreto-Naehrig curves was very effective and resulted in a 70% execution time reduction.
The conclusion drawn from this data was that one can predict how much can be gained from hardware acceleration from a callgrind graph. The best way to get an execution time improvement is to find a function in the callgrind graph with a great execution time that can be replaced by a single hardware accelerated operation.

Two of the best configurations in the execution time measurements were selected for the energy consumption tests. The pairing on the Barreto-Naehrig curve used on average 0.127 Joule, on the supersingular with a higher security level it takes 0.293 Joule.
While some variance was detected in the measurements, both caused by testing on different days and on different instances of RE-Mote, all measurements fell within 1% of their mean.

Finally, some tests were performed to measure the resource usage when communicating inputs or output of a pairing operation. These showed that the resource usage to communicate one operand is generally less than 10% of the computation. This value can however easily rise above this percentage when all operands are communicated. It is also noted how these values are very dependent on configuration details like the algorithm in which pairings are used, the communication protocol and the network topology.

One can use these results to determine if it is feasible to use pairing-based cryptography on a constrained device for their application. In general, this work has shown that there can be situations where it is more resource efficient to perform a pairing operation locally on a constrained device than to communicate the operands through a network to have a third party compute the result.

7 Future Work

Throughout this report, several ways have been discussed in which the project could be improved in the future. This section elaborates on these possible ways to continue this work in the future.

An interesting future research path could involve analysing more optimistic scenarios. As shown by Hajny et al. [17], the PBC is a relatively inefficient library to perform pairing-based cryptography computations. An alternative direction could be to optimise PBC for IoT devices, by minimising dynamic memory allocations and simplifying the code base to reduce size. The expected outcome would be a more positive picture of pairing-based cryptography on constrained devices. Hopefully this also allows pairings at higher levels of security, offering more security for the future.

This research can become significantly more valuable by presenting more data. Testing different devices, possibly supporting different types of hardware acceleration, different types of curves at different sizes, different communication protocols, etc. This all helps to provide a more comprehensive overview of the challenges and possibilities to use pairing-based cryptography on constrained devices. In turn, it will be easier to use this research to predict performance for specific applications using more complex cryptographic algorithms.

The final proposed work for the future is to improve the measurement setup. The first priority would be to make the setup more accurate and faster. One way to do this is to replace the Arduino Due by a different device that can communicate over I2C. Alternatively, one may build a different setup using an oscilloscope and a shunt resistor. The desired outcome is to measure with higher frequency, increasing the likelihood of measuring short bursts of power draw, and measuring them at a higher resolution.
Secondly, the measurement setup would benefit from more and improved automation. The many manual steps in measuring introduce many possible points of human error. Additionally, the automated parts, such as the script to extract the useful part of the measurement, could be improved to require less tuning.
Finally, an improved measurement setup should be used to identify the variances in measurements. By controlling the maximum amount variables, the goal would be to identify which influence the execution time and power draw the most, such that tests can correct for these effects.

Acronyms

ABE attribute-based encryption. 8

ADC analog to digital converter. 19

BN Barreto-Naehrig. 2, 8, 11, 12, 15, 17, 20–24, 26–29, 33, 35, 45

CDH computational Diffie-Hellman problem. 7

CISC Complex Instruction Set Computer. 9

CoAP constrained application protocol. 29

DDH decisional Diffie-Hellman problem. 7

FPGA field-programmable gate array. 12

GMP GNU multiple precision arithmetic library. 16

HWA hardware acceleration. 13, 18, 24, 26–28

IBE identity-based encryption. 8

IoT Internet of Things. 1, 5, 9–11, 13, 15, 16, 19

MNT Miyaji-Nakabayashi-Takano. 8, 12

OS operating system. 14

PBC pairing-based cryptography. 1, 2, 5, 7–9, 11, 12, 15–17, 20, 22, 24, 26, 35, 37

RISC Reduced Instruction Set Computer. 9

SS supersingular. 2, 8, 12, 15, 17, 20–29, 35, 45, 46

WSN wireless sensor network. 9, 10, 12, 30

References

[1] Mohammad Y Al-Shorman, Majd M Al-Kofahi, and Osameh M Al-Kofahi. A practical microwatt-meter for electrical energy measurement in programmable devices. *Measurement and Control*, 51(9-10):383–395, aug 2018.

[2] Ahmad Ali, Yu Ming, Sagnik Chakraborty, and Saima Iram. A comprehensive survey on real-time applications of WSN. *Future Internet*, 9(4):77, nov 2017.

[3] Emmanuel Baccelli, Cenk Gundogan, Oliver Hahm, Peter Kietzmann, Martine S. Lenders, Hauke Petersen, Kaspar Schleiser, Thomas C. Schmidt, and Matthias Wahlisch. RIOT: An open source operating system for low-end embedded devices in the IoT. *IEEE Internet of Things Journal*, 5(6):4428–4440, dec 2018.

[4] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography*, pages 319–331. Springer Berlin Heidelberg, 2006.

[5] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. IEEE, may 2007.

[6] Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. *SIAM Journal on Computing*, 32(3):586–615, jan 2003.

[7] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *Lecture Notes in Computer Science*, pages 416–432. Springer Berlin Heidelberg, 2003.

[8] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *Journal of Cryptology*, 17(4):297–319, jul 2004.

[9] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In *Proceedings of the 11th ACM conference on Computer and communications security - CCS '04*. ACM Press, 2004.

[10] Sébastien Canard, Nicolas Desmoulins, Julien Devigne, and Jacques Traoré. On the implementation of a pairing-based cryptographic protocol in a constrained device. In *Pairing-Based Cryptography – Pairing 2012*, pages 210–217. Springer Berlin Heidelberg, 2013.

[11] Ratna Dutta, Rana Barua, and Palash Sarkar. Pairing-based cryptographic protocols: A survey. Cryptology ePrint Archive, Report 2004/064, 2004.

[12] Omid Mahdi Ebadati E., Seyed Mahdi Sadat Rasoul, Kaebeh Yaeghoobi, and Faezeh Hadadi. Sensing, communication with efficient and sustainable energy: An IoT framework for smart cities. In *Smart Technologies for Energy and Environmental Sustainability*, pages 53–86. Springer International Publishing, nov 2021.

[13] David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, jun 2009.

[14] Gerhard Frey and Hans-Georg Ruck. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 62(206):865, apr 1994.

[15] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, sep 2008.

[16] Benedetto Girgenti, Pericle Perazzo, Carlo Vallati, Francesca Righetti, Gianluca Dini, and Giuseppe Anastasi. On the feasibility of attribute-based encryption on constrained IoT devices for smart systems. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, jun 2019.

[17] Jan Hajny, Petr Dzurenda, Sara Ricci, Lukas Malina, and Kamil Vrba. Performance analysis of pairing-based elliptic curve cryptography on constrained devices. In *2018 10th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. IEEE, nov 2018.

[18] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2004.

[19] IEEE 802.15 WPAN Task Group 4. IEEE standard for low-rate wireless networks. Technical report, IEEE, 2020.

[20] ISO. Cryptographic techniques based on elliptic curves - part 5: Elliptic curve generation. Standard, International Organization for Standardization, 2022.

[21] Antoine Joux. A one round protocol for tripartite diffie–hellman. *Journal of Cryptology*, 17(4):263–276, jun 2004.

[22] Diptendu M. Kar and Indrajit Ray. Systematization of knowledge and implementation: Short identity-based signatures. August 2019.

[23] Arjen K. Lenstra. Key lengths contribution to the handbook of information security. 2010.

[24] Ben Lynn. *On the implementation of Pairing-Based Cryptosystems*. PhD thesis, Stanford University, June 2007.

[25] Dennis Meffert. Bilinear pairings in cryptography. Online, May 2009. `https://www.math.ru.nl/~bosma/Students/MScThesis_DennisMeffert.pdf`.

[26] Alfred Menezes. An introduction to pairing-based cryptography. October 2013.

[27] Alfred Menezes, Scott Vanstone, and Tatsuaki Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing - STOC '91*. ACM Press, 1991.

[28] George Oikonomou, Simon Duquennoy, Atis Elsts, Joakim Eriksson, Yasuyuki Tanaka, and Nicolas Tsiftes. The Contiki-NG open source operating system for next generation IoT devices. *SoftwareX*, 18:101089, jun 2022.

[29] Aleksandr Ometov, Pavel Masek, Lukas Malina, Roman Florea, Jiri Hosek, Sergey Andreev, Jan Hajny, Jussi Niutanen, and Yevgeni Koucheryavy. Feasibility characterization of cryptographic primitives for constrained (wearable) IoT devices. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. IEEE, mar 2016.

[30] D. Page, N. Smart, and F. Vercauteren. A comparison of MNT curves and supersingular curves. *Applicable Algebra in Engineering, Communication and Computing*, 17:379–392, 10 2006.

[31] Pericle Perazzo, Francesca Righetti, Michele La Manna, and Carlo Vallati. Performance evaluation of attribute-based encryption on constrained IoT devices. *Computer Communications*, 170:151–163, mar 2021.

[32] Archana Ramachandran, Zhibin Zhou, and Dijiang Huang. Computing cryptographic algorithms in portable and embedded devices. In *2007 IEEE International Conference on Portable Information Devices*. IEEE, may 2007.

[33] Y. Sakemi, T. Kobayashi, T. Saito, and R. Wahby. Pairing-friendly curves, January 2022.

[34] Ahmad Salman, William Diehl, and Jens-Peter Kaps. A light-weight hardware/software co-design for pairing-based cryptography with low power and energy consumption. In *2017 International Conference on Field Programmable Technology (ICFPT)*, pages 235–238, 2017.

[35] Sriram Sankaran. Lightweight security framework for IoTs using identity based cryptography. In *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, sep 2016.

[36] Savio Sciancalepore, Gabriele Oligeri, and Roberto Di Pietro. Strength of crowd (SOC)—defeating a reactive jammer in IoT with decoy messages. *Sensors*, 18(10):3492, oct 2018.

[37] Michael Scott and Paulo S.L.M. Barreto. Generating more MNT elliptic curves. *Designs, Codes and Cryptography*, 28, February 2006.

[38] Pallavi Sethi and Smruti R. Sarangi. Internet of things: Architectures, protocols, and applications. *Journal of Electrical and Computer Engineering*, 2017:1–25, 2017.

[39] Saurabh Singh, Pradip Kumar Sharma, Seo Yeon Moon, and Jong Hyuk Park. Advanced lightweight encryption algorithms for IoT devices: survey, challenges and solutions. *Journal of Ambient Intelligence and Humanized Computing*, may 2017.

[40] Piotr Szczechowiak, Anton Kargl, Michael Scott, and Martin Collier. On the application of pairing based cryptography to wireless sensor networks. In *Proceedings of the second ACM conference on Wireless network security - WiSec '09*. ACM Press, 2009.

[41] Texas Instruments. CC2538 System-on-Chip Solution for 2.4-GHz IEEE 802.15.4 and ZigBee®/ZigBee IP® Applications. 2013.

[42] Texas Instruments. INA219 zero-drift, bidirection current/power monitor with I2C interface. 2015.

[43] Valgrind Developers. Callgrind: a call-graph generating cache and branch prediction profiler. Online `https://valgrind.org/docs/manual/cl-manual.html`. Last accessed May 22, 2022.

[44] Xuanxia Yao, Zhi Chen, and Ye Tian. A lightweight attribute-based encryption scheme for the internet of things. *Future Generation Computer Systems*, 49:104–112, aug 2015.

[45] Joseph Yiu. ARM Cortex-M for beginners: An overview of the ARM Cortex-M processor family and comparison. Technical report, ARM, March 2017.

[46] Binbin Yu and Hongtu Li. Anonymous authentication key agreement scheme with pairing-based cryptography for home-based multi-sensor internet of things. *International Journal of Distributed Sensor Networks*, 15(9):155014771987937, sep 2019.

[47] Lu Zhou, Chunhua Su, and Kuo-Hui Yeh. A lightweight cryptographic protocol with certificateless signature for the internet of things. *ACM Transactions on Embedded Computing Systems*, 18(3):1–10, jun 2019.

A Curve parameters

A.1 BN 53

Generated using `pb_param_init_f_gen(parameters, 53)`.

```
q      = 672220231350823
r      = 672220205417377
b      = 111399333710995
beta   = 509999402509525
alpha0 = 397313476779944
alpha1 = 307533454640401
```

A.2 SS 318

Generated using `pbc_param_init_a_gen(parameters, 160, 318)`.

```
q      = 384284696635097971885434455202028844552941311824
         399003835550838454995940390972850730352170371 47
h      = 525876268049579567842409815013488694543267436708
r      = 730751167114595186142829002853739519958614802431
exp1   = 128
exp2   = 159
sign0  = −1
sign1  = 1
```

A.3 SS 512

Generated using `pbc_param_init_a_gen(parameters, 160, 512)`.

```
q      = 255924955157650670516423004233366706214305385605 50
         086238294375266242321140927190193065045904197241 85
         8828084036025639
h      = 350221920551575661252522731512754917864310999788 20
         680852024212634920
r      = 730750818665451621361119245571504901405976559617
exp1   = 107
exp2   = 159
sign0  = 1
sign1  = 1
```

A.4 SS 700

Generated using `pbc_param_init_a_gen(parameters, 160, 700)`.

```
q     = 46627891161150428916710649802834087853435010026153
        34469384822955925632547299171660569484932170229366
        46347107388502055095979954873341076120197804740677
        33078422696160069748821992596026559519228907668620
        8917577471
h     = 63808191781525246219613177528601056410312484691950
        86362893712600021468846590063875192757132991704012
        02535932901477366315385843058236108613060906988730
        590111997696
r     = 730750862221594424981965739670091261094297337857
exp1  = 135
exp2  = 159
sign0 = 1
sign1 = 1
```

B Detailed callgrind graphs

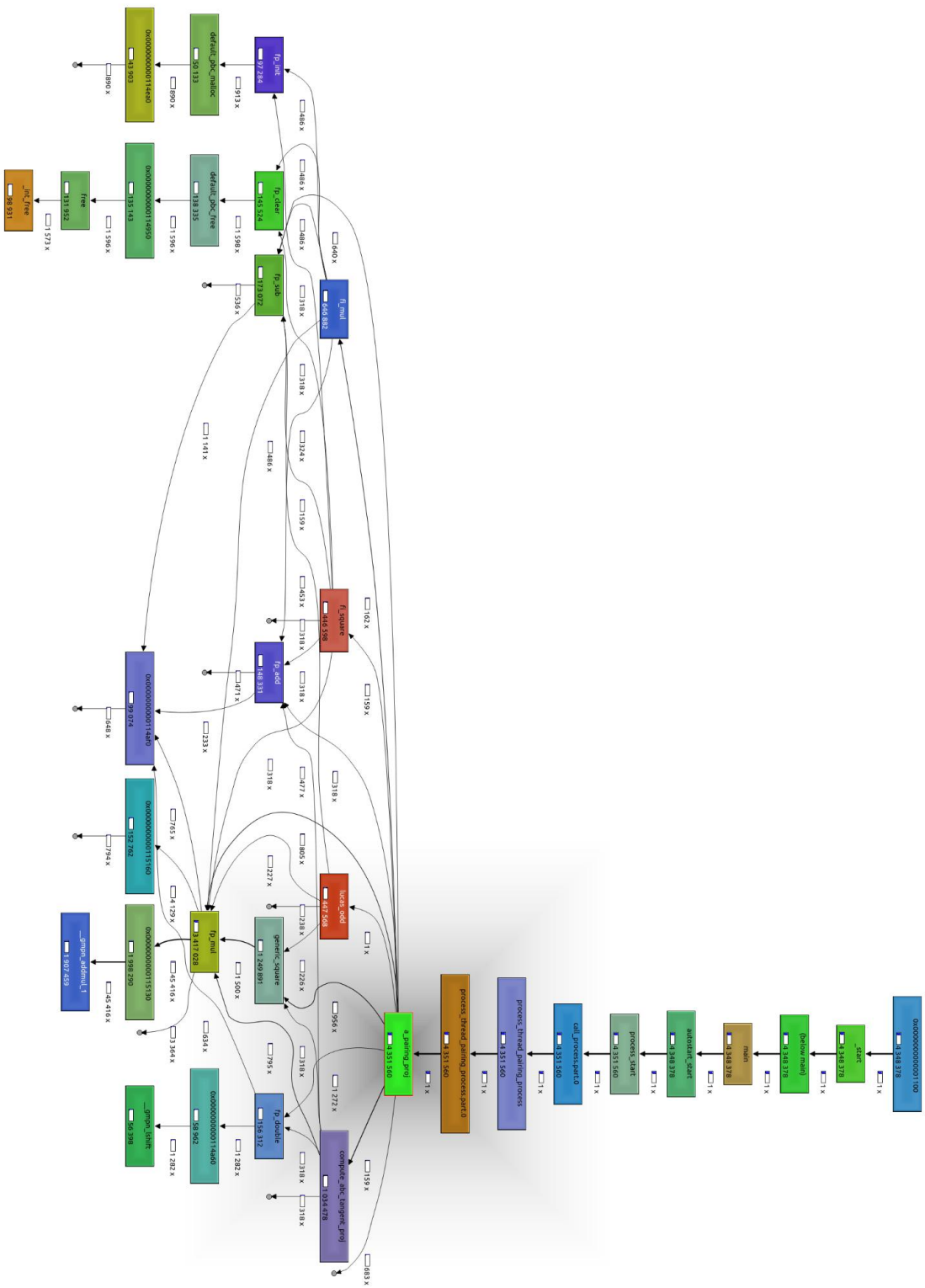

Figure 13: More detailed and larger version of figure 4

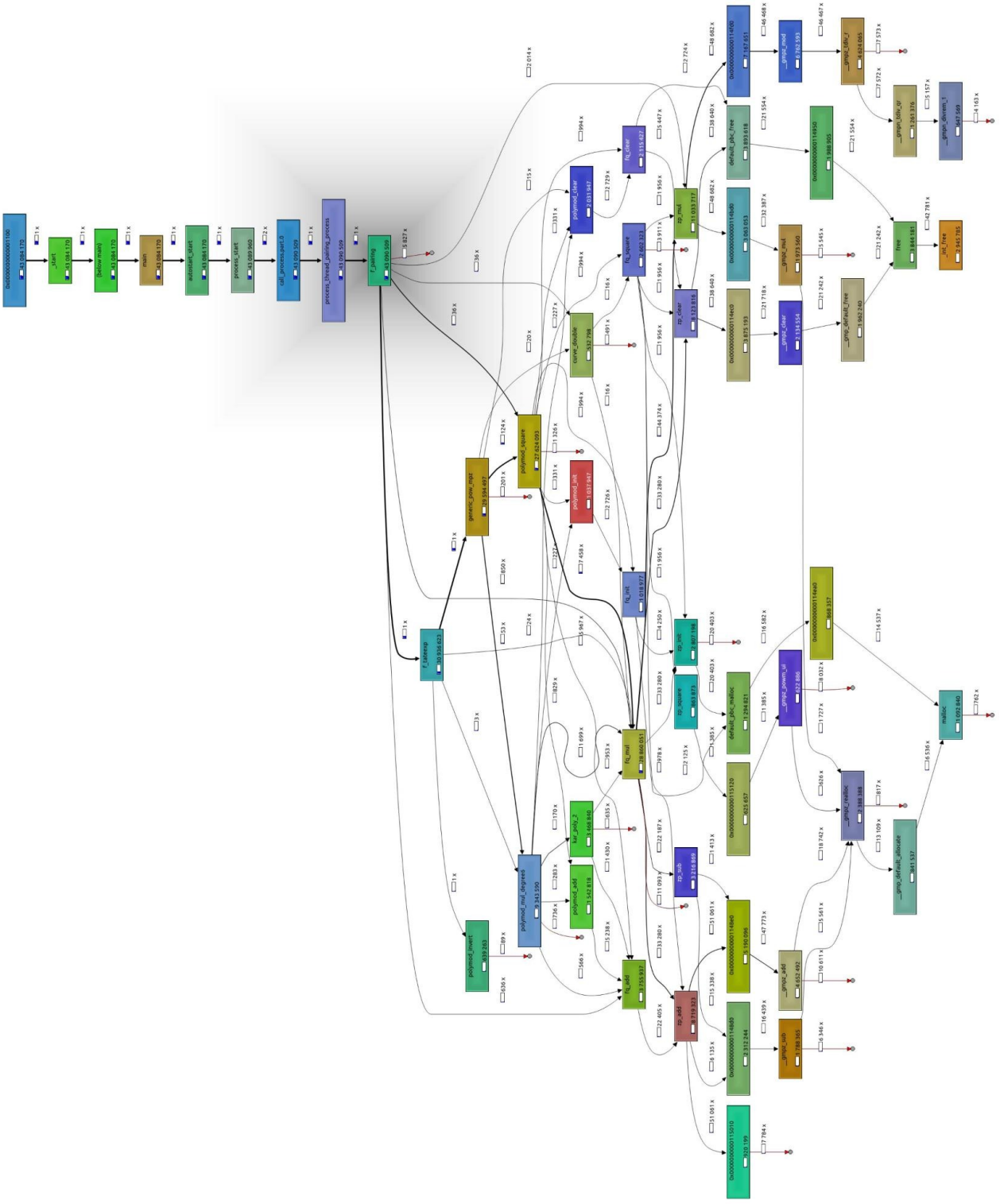

Figure 14: More detailed and larger version of figure 5