Master of Science HES-SO in Engineering
Av. de Provence 6
CH-1007 Lausanne

Master of Science HES-SO in Engineering

Orientation: Information and Communication Technologies (ICT)

NEW METHODS FOR TRANSACTIONS IN BLOCKCHAIN SYSTEMS

Author:
Joël Gugger

Under the direction of:
Prof. Alexandre Karlov
HEIG-VD

External expert:
firstname lastname
lab

Lausanne, HES-SO//Master, January 26, 2018

Information about this report

Contact information

Author: JoëlGugger
MSE Student
HES-SO//Master
Switzerland
Email: *joel.gugger@master.hes-so.ch*

Declaration of honor

I, undersigned, Joël Gugger, hereby declare that the work submitted is the result of a personal work. I certify that I have not resorted to plagiarism or other forms of fraud. All sources of information used and the author quotes were clearly mentioned.

Place, date: ________________________________

Signature: ________________________________

Validation

Accepted by the HES-SO//Master (Switzerland, Lausanne) on a proposal from:

Prof. Alexandre Karlov, Thesis project advisor
firstname lastname, lab, Main expert

Place, date: ________________________________

Prof. Alexandre Karlov	Prof. Fariba Moghaddam Bützberger
Advisor	Dean, HES-SO//Master

Acknowledgments

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Preface

A preface is not mandatory. It would typically be written by some other person (eg your thesis director).
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.
Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Lausanne, 12 Mars 2011 T. D.

Abstract

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Keywords: keyword1, keyword2, keyword3

Contents

1 Introduction

What is Bitcoin? why do we need it? [1, 2, 3, 4, 5]

ECDSA is the signature scheme used by Bitcoin to sign transactions. A standard transaction is constituted of a single signature corresponding to the address where the Bitcoins come from. But sometimes we need more complex management for locking funds. To address the limitation of a single signature, Bitcoin introduced a new OP_CODE named CHECKMULTISIG with a new standard script. With this standard script, it is now possible to spend Bitcoin to an address that requires a minimum of m signatures in n authorised signatories and extend the capability of Bitcoin to lock funds in a more complex way.

However, some issues appear. The way the script works requires exposing all the public keys when an output is signed and this increases the transaction size enormously, which implies bigger fees. All the signatures are, obviously, present with the public keys in the transaction script, which implies that we can know which public keys signed the transaction. And there is some limitation, due to the script size limit, the maximum number of authorised signatories is 15. All these limitations mean that we cannot imagine a complex organization nor structure with the multi-signature script for the moment.

To address this limitation, a group of researchers published a first paper in 2015 and a second one in 2016 describing the way to achieve a threshold scheme with DSA and ECDSA. Today, there is no well-known implementation ready for production purposes even though industries need it. The principal purpose of this thesis is to provide a clear, well documented C library, based on the internal ECDSA library present in bitcoin-core.

The largest challenge in Bitcoin for the coming years is scalability. Currently, Bitcoin enforces a block-size limit which is equivalent to only some transactions per second on the network. This is not sufficient in comparison to big payment infrastructures such as VISA, which allows tens of thousands of transactions per second and even more in peak times such as Christmas. To address this, some proposals modifying the transaction structure (like SegWit), some proposals modifying the block-size limit (such as SegWit2x) and others creating a second layer based on top of the Bitcoin protocol (like Lightning Network) exist. In the same idea of the Lightning Network, Bity is working on an implementation of a one-way payment channel. A one-way payment channel allows two parties to transact over the blockchain while minimizing the number of transactions needed on the blockchain in a secure and trustless way. This kind of channel needs multi-signature addresses which might be improved with the threshold scheme. The second part of the thesis is to co-write the channel white paper and add a chapter of how to improve it with the threshold scheme (better privacy, cheaper transaction, less limitations).

2 Bitcoin, a peer-to-peer payment network

Concepts Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Contents

2.1 Actors

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

type of nodes in the peer-to-peer network `https://github.com/bitcoinbook/bitcoinbook/blob/second_edition/ch08.asciidoc`

2.1.1 Users

2.1.2 Miners

2.1.3 Developpers

2.2 Blockchain

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2.2.1 Public ledger

2.2.2 Speed

Mining difficulty

2.3 Transaction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2.3.1 Scripting language

Locking script, unlocking script

2.3.2 Transaction Fees

2.4 Scalability of Bitcoin

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa.

Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2.4.1 On-chain improvments

2.4.2 Layer-two applications

3 Payment channels, a micro-transaction network

Concept Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Contents

3.1 Types of payment channel

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

3.2 Our one-way channel

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

3.3 Optimizing channels

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

4 Threshold optimal ECDSA scheme

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Contents

4.1 Elliptic Curve

Bitcoin use elliptic curve cryptography for securing Bitcoin transaction. The curve secp256k1, proposed by the Standards for Efficient Cryptography (SEC), is used.

4.1.1 Secp256k1 curve

The curve E is define in $\mathbb{F}_p$ with a Koblitz curve

$$\begin{aligned} y^2 &= x^3 + ax + b \\ y^2 &= x^3 + 7 \end{aligned}$$

4.1.2 Point addition

With two distinct point P and Q on the curve E, the resulting point of the addition is the inverse point, $(x, -y)$, of the intersection point with a straight line between P and Q.

$$\begin{aligned} P + Q &= R \\ (x_p, y_p) + (x_q, y_q) &= (x_r, y_r) \\ x_r &\equiv \lambda^2 - x_p - x_q \pmod p \\ y_r &\equiv \lambda(x_p - x_r) - y_p \pmod p \\ \lambda &= \frac{y_q - y_p}{x_q - x_p} \\ &\equiv (y_q - y_p)(x_q - x_p)^{-1} \pmod p \end{aligned}$$

4.1.3 Point doubling

For P and Q equal, the formula is similar, the tangent to the curve E at point P is used to determine R.

$$\begin{aligned} P + P &= R \\ (x_p, y_p) + (x_p, y_p) &= (x_r, y_r) \\ x_r &\equiv \lambda^2 - 2x_p \pmod p \\ y_r &\equiv \lambda(x_p - x_r) - y_p \pmod p \\ \lambda &= \frac{3x_p^2 + a}{2y_p} \\ &\equiv (3x_p^2 + a)(2y_p)^{-1} \pmod p \end{aligned}$$

4.1.4 Point multiplication

A point P can be multiply by a scalar d. The straightforward way of computing a point multiplication is through repeated addition. However, this is a fully exponential approach to computing the multiplication. The simplest method is the double-and-add method. To compute dP, start with the binary representation for $d : d = d_0 + 2d_1 + 2^2d_2 + \cdots + 2^m d_m$, where $[d_0 \dots d_m] \in 0, 1$

$$dP = P_1 + P_2 + \cdots + P_d$$

4.2 Threshold scheme

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

4.2.1 Adapting zero-knowledge proofs to ECDSA

Initialy the proofs have been designed for the DSA architecture, so the values tested in the proofs are values in $\mathbb{Z}_q$. These values are used to create a challenge e with two hash function (one different per proof.) For ECDSA some of these values are now points and some equation need to be adapted. Points are serialized in long form of 65 bytes, starting with `0x04` and two 32 bytes number for the coordinate x and y.

The first zero-knowledge proof Π is created by Alice to prove to Bob that she act correctly and have encrypted coherent data with Paillier encryption.

$$
\begin{array}{|ll|}
\hline
\alpha \xleftarrow{R} \mathbb{Z}_{n^3} & \delta \xleftarrow{R} \mathbb{Z}_{n^3} \\
\beta \xleftarrow{R} \mathbb{Z}_N^* & \mu \xleftarrow{R} \mathbb{Z}_N^* \\
\gamma \xleftarrow{R} \mathbb{Z}_{n^3\tilde{N}} & \nu \xleftarrow{R} \mathbb{Z}_{n^3\tilde{N}} \\
\rho_1 \xleftarrow{R} \mathbb{Z}_{n\tilde{N}} & \rho_2 \xleftarrow{R} \mathbb{Z}_{n\tilde{N}} \\
 & \rho_3 \xleftarrow{R} \mathbb{Z}_n \\
 & \epsilon \xleftarrow{R} \mathbb{Z}_n \\
 & \\
z_1 \leftarrow (h_1)^{x_1}(h_2)^{\rho_1} \mod \tilde{N} & z_2 \leftarrow (h_1)^{x_2}(h_2)^{\rho_2} \mod \tilde{N} \\
u_1 \leftarrow c \cdot \alpha & y \leftarrow d \cdot (x_2 + \rho_3) \\
u_2 \leftarrow g^{\alpha}\beta^N \mod N^2 & v_1 \leftarrow d \cdot (\delta + \epsilon) \\
u_3 \leftarrow (h_1)^{\alpha}(h_2)^{\gamma} \mod \tilde{N} & v_2 \leftarrow w_2 \cdot \alpha + d \cdot \epsilon \\
 & v_3 \leftarrow g^{\delta}\mu^N \mod N^2 \\
 & v_4 \leftarrow (h_1)^{\delta}(h_2)^{\nu} \mod \tilde{N} \\
 & \\
\multicolumn{2}{|l|}{e \leftarrow \mathtt{hash}(c, w_1, d, w_2, m_1, m_2, z_1, u_1, u_2, u_3, z_2, y, v_1, v_2, v_3, v_4)} \\
 & \\
s_1 \leftarrow ex_1 + \alpha & t_1 \leftarrow ex_2 + \delta \\
s_2 \leftarrow (r_1)^e\beta \mod N & t_2 \leftarrow e\rho_3 + \epsilon \mod n \\
s_3 \leftarrow e\rho_1 + \gamma & t_3 \leftarrow (r_2)^e\mu \mod N^2 \\
 & t_4 \leftarrow e\rho_2 + \nu \\
 & \\
\multicolumn{2}{|c|}{\Pi \leftarrow \langle z_1, z_2, y, e, s_1, s_2, s_3, t_1, t_2, t_3, t_4 \rangle} \\
\hline
\end{array}
$$

Figure 4.1 Adaptation of Π *to ECDSA*

$$
\begin{array}{|ll|}
\hline
\multicolumn{2}{|c|}{\langle z_1, z_2, y, e, s_1, s_2, s_3, t_1, t_2, t_3, t_4 \rangle \leftarrow \Pi} \\
 & \\
\text{Verify } s_1, t_1 \in \mathbb{Z}_{n^3} & v_1 \leftarrow d \cdot (t_1 + t_2) + y \cdot (-e) \\
u_1 \leftarrow c \cdot s_1 + w_1 \cdot (-e) & v_2 \leftarrow w_2 \cdot s_1 + d \cdot t_2 + y \cdot (-e) \\
u_2 \leftarrow g^{s_1}(s_2)^N(m_1)^{-e} \mod N^2 & v_3 \leftarrow g^{t_1}(t_3)^N(m_2)^{-e} \mod N^2 \\
u_3 \leftarrow (h_1)^{s_1}(h_2)^{s_3}(z_1)^{-e} \mod \tilde{N} & v_4 \leftarrow (h_1)^{t_1}(h_2)^{t_4}(z_2)^{-e} \mod \tilde{N} \\
 & \\
\multicolumn{2}{|c|}{\text{Verify } e = \mathtt{hash}(c, w_1, d, w_2, m_1, m_2, z_1, u_1, u_2, u_3, z_2, y, v_1, v_2, v_3, v_4)} \\
\hline
\end{array}
$$

Figure 4.2 Adaptation of Π *verification to ECDSA*

Second zero-knowledge proof created by Bob to prove to Alice that he acted correctly according the protocol.

$$\alpha \xleftarrow{R} \mathbb{Z}_{n^3} \qquad \delta \xleftarrow{R} \mathbb{Z}_{n^3}$$
$$\beta \xleftarrow{R} \mathbb{Z}^*_{N'} \qquad \mu \xleftarrow{R} \mathbb{Z}^*_{N}$$
$$\gamma \xleftarrow{R} \mathbb{Z}_{n^3\tilde{N}} \qquad \nu \xleftarrow{R} \mathbb{Z}_{n^3\tilde{N}}$$
$$\rho_1 \xleftarrow{R} \mathbb{Z}_{n\tilde{N}} \qquad \rho_2 \xleftarrow{R} \mathbb{Z}_{n\tilde{N}}$$
$$\rho_3 \xleftarrow{R} \mathbb{Z}_{n}$$
$$\rho_4 \xleftarrow{R} \mathbb{Z}_{n^5\tilde{N}}$$
$$\epsilon \xleftarrow{R} \mathbb{Z}_{n}$$
$$\sigma \xleftarrow{R} \mathbb{Z}_{n^7}$$
$$\tau \xleftarrow{R} \mathbb{Z}_{n^7\tilde{N}}$$

$$z_1 \leftarrow (h_1)^{x_1}(h_2)^{\rho_1} \mod \tilde{N} \qquad z_2 \leftarrow (h_1)^{x_2}(h_2)^{\rho_2} \mod \tilde{N}$$
$$u_1 \leftarrow c \cdot \alpha \qquad y \leftarrow d \cdot (x_2 + \rho_3)$$
$$u_2 \leftarrow (g')^{\alpha}\beta^{N'} \mod (N')^2 \qquad v_1 \leftarrow d \cdot (\delta + \epsilon)$$
$$u_3 \leftarrow (h_1)^{\alpha}(h_2)^{\gamma} \mod \tilde{N} \qquad v_2 \leftarrow w_2 \cdot \alpha + d \cdot \epsilon$$
$$v_3 \leftarrow (m_3)^{\alpha}(m_4)^{\delta} g^{n\sigma}\mu^{N} \mod N^2$$
$$v_4 \leftarrow (h_1)^{\delta}(h_2)^{\nu} \mod \tilde{N}$$
$$z_3 \leftarrow (h_1)^{x_3}(h_2)^{\rho_4} \mod \tilde{N}$$
$$v_5 \leftarrow (h_1)^{\sigma}(h_2)^{\tau} \mod \tilde{N}$$

$$e \leftarrow \texttt{hash'}(c, w_1, d, w_2, m_1, m_2, z_1, u_1, u_2, u_3, z_2, z_3, y, v_1, v_2, v_3, v_4, v_5)$$

$$s_1 \leftarrow ex_1 + \alpha \qquad t_1 \leftarrow ex_2 + \delta$$
$$s_2 \leftarrow (r_1)^e\beta \mod N' \qquad t_2 \leftarrow e\rho_3 + \epsilon \mod n$$
$$s_3 \leftarrow e\rho_1 + \gamma \qquad t_3 \leftarrow (r_2)^e\mu \mod N$$
$$s_4 \leftarrow ex_1x_4 + \alpha \qquad t_4 \leftarrow e\rho_2 + \nu$$
$$t_5 \leftarrow ex_3 + \sigma$$
$$t_6 \leftarrow e\rho_4 + \tau$$
$$t_7 \leftarrow ex_2x_5 + \delta$$

$$\Pi' \leftarrow \langle z_1, z_2, z_3, y, e, s_1, s_2, s_3, s_4, t_1, t_2, t_3, t_4, t_5, t_6, t_7 \rangle$$

Figure 4.3 Adaptation of Π' to ECDSA

$$\langle z_1, z_2, z_3, y, e, s_1, s_2, s_3, s_4, t_1, t_2, t_3, t_4, t_5, t_6, t_7 \rangle \leftarrow \Pi'$$

Verify $s_1, t_1 \in \mathbb{Z}_{n^3}$	$v_1 \leftarrow d \cdot (t_1 + t_2) + y \cdot (-e)$
Verify $t_5 \in \mathbb{Z}_{n^7}$	$v_2 \leftarrow w_2 \cdot s_1 + d \cdot t_2 + y \cdot (-e)$
$u_1 \leftarrow c \cdot s_1 + w_1 \cdot (-e)$	$v_3 \leftarrow (m_3)^{s_4}(m_4)^{t_7} g^{nt_5}(t_3)^N(m_2)^{-e} \mod N^2$
$u_2 \leftarrow (g')^{s_1}(s_2)^{N'}(m_1)^{-e} \mod (N')^2$	$v_4 \leftarrow (h_1)^{t_1}(h_2)^{t_4}(z_2)^{-e} \mod \tilde{N}$
$u_3 \leftarrow (h_1)^{s_1}(h_2)^{s_3}(z_1)^{-e} \mod \tilde{N}$	$v_5 \leftarrow (h_1)^{t_5}(h_2)^{t_6}(z_3)^{-e} \mod \tilde{N}$

Verify $e = \texttt{hash'}(c, w_1, d, w_2, m_1, m_2, z_1, u_1, u_2, u_3, z_2, z_3, y, v_1, v_2, v_3, v_4, v_5)$

Figure 4.4 Adaptation of Π' verification to ECDSA

If $x_1 = z_2$, $x_2 = x_2 z_2$, $x_3 = c$, and $m_2 = \mu$ with $\mu = (\alpha)^{m'x_1}(\zeta)^{r'x_2} g^{nx_3}(r_2)^N$, then the equation v_3 in the original paper doesn't work. The result in the verification proccess Π' need to match $v_3 \leftarrow (m_3)^{\alpha}(m_4)^{\delta} g^{n\sigma}\mu^N \mod N^2$. The original equation proposed $v_3 \leftarrow (m_3)^{s_1}(m_4)^{t_1} g^{nt_5}(t_3)^N(m_2)^{-e} \mod N^2$ doesn't include m' and r' present in μ, so m_2 cannot be used correctly as showed next.

$$\begin{aligned}
v_3 &\equiv (m_3)^{s_1}(m_4)^{t_1} g^{nt_5}(t_3)^N(m_2)^{-e} \pmod{N^2} \\
&\equiv (m_3)^{ex_1+\alpha}(m_4)^{ex_2+\beta} g^{n(ex_3+\sigma)}\left((r_2)^e\mu\right)^N\left((m_3)^{m'x_1}(m_4)^{r'x_2} g^{nx_3}(r_2)^N\right)^{-e} \\
&\equiv (m_3)^{ex_1+\alpha}(m_4)^{ex_2+\beta} g^{n(ex_3+\sigma)}(r_2)^{eN}\mu^N(m_3)^{-em'x_1}(m_4)^{-er'x_2} g^{-enx_3}(r_2)^{-eN} \\
&\equiv (m_3)^{ex_1+\alpha-em'x_1}(m_4)^{ex_2+\beta-er'x_2} g^{enx_3+n\sigma-enx_3}(r_2)^{eN-eN}\mu^N \\
&\equiv (m_3)^{ex_1+\alpha-em'x_1}(m_4)^{ex_2+\beta-er'x_2} g^{n\sigma}\mu^N
\end{aligned}$$

The equation v_3 needs to be adapted to include $x_4 = m'$ and $x_5 = r'$ (m' and r' cannot be include directly in x_1 and x_2 without breaking other equations.) Two new parameters $s_4 \leftarrow ex_1x_4 + \alpha$ and $t_7 \leftarrow ex_2x_5 + \delta$ are added into the proof to correct the equation.

$$\begin{aligned}
v_3 &\equiv (m_3)^{s_4}(m_4)^{t_7} g^{nt_5}(t_3)^N(m_2)^{-e} \pmod{N^2} \\
&\equiv (m_3)^{ex_1x_4+\alpha}(m_4)^{ex_2x_5+\beta} g^{n(ex_3+\sigma)}\left((r_2)^e\mu\right)^N\left((m_3)^{x_1x_4}(m_4)^{x_2x_5} g^{nx_3}(r_2)^N\right)^{-e} \\
&\equiv (m_3)^{ex_1x_4+\alpha}(m_4)^{ex_2x_5+\beta} g^{n(ex_3+\sigma)}(r_2)^{eN}\mu^N(m_3)^{-ex_1x_4}(m_4)^{-ex_2x_5} g^{-enx_3}(r_2)^{-eN} \\
&\equiv (m_3)^{ex_1x_4+\alpha-ex_1x_4}(m_4)^{ex_2x_5+\beta-ex_2x_5} g^{enx_3+n\sigma-enx_3}(r_2)^{eN-eN}\mu^N \\
&\equiv (m_3)^{\alpha}(m_4)^{\beta} g^{n\sigma}\mu^N
\end{aligned}$$

4.3 Threshold Hierarchical Determinitic Wallets

Hierarchical deterministic wallets are sophisticated wallets in wich fresh keys can be generated from a previous key. Adapting hierarchical deterministic wallets with a threshold scheme can be achieve by sharing the private key additively:

$$
\begin{aligned}
pk_i &= sk_i \cdot G \\
sk_{mas} &= \sum_{i=1}^{s} sk_i \bmod n \\
pk_{mas} &= \left[\sum_{i=1}^{s} sk_i \bmod n \right] \cdot G \\
&= \sum_{i=1}^{s} (sk_i \cdot G) = \sum_{i=1}^{s} pk_i
\end{aligned}
$$

or multiplicatively:

$$
\begin{aligned}
sk_{mas} &= \prod_{i=1}^{s} sk_i \bmod n \\
pk_{mas} &= \left[\prod_{i=1}^{s} sk_i \bmod n \right] \cdot G \\
&= (((G \cdot sk_1) \cdot sk_2) \ldots) \cdot sk_i
\end{aligned}
$$

In the additive case, the master public key pk_{mas} is also the sum of all the public points pk_i, which means that if each one publish his own public share point, every one can compute the master public key.

4.3.1 Private parent key to private child key

The function CKDpriv compute a child extended private key from the parent extended private key. The derivatioin can be *hardened*. This proposal differ from the BIP32 standard in the chain derivation proccess. The threshold scheme require the same chain on all participants, so the proccess cannot rely on the private key share of any participant.

$$
\begin{aligned}
f(l) &= \begin{cases} \text{HMAC-SHA256}(c_{par}, \text{0x00} \mid\mid \text{ser}_{256}(k_{par}) \mid\mid \text{ser}_{32}(i)) & \text{if } i \geq 2^{31} \\ \text{HMAC-SHA256}(c_{par}, \text{ser}_p(\text{point}(k_{par})) \mid\mid \text{ser}_{32}(i)) & \text{if } i < 2^{31} \end{cases} \\
k_i &\equiv l \cdot k_{par} \pmod n
\end{aligned}
$$

4.3.2 Public parent key to public child key

The function CKDpub compute a child extend public key from the parent extended public key. It is worth noting than it is not possible to compute an *hardened* derivation without the parent private key.

$$
\begin{aligned}
f(l) &= \begin{cases} \text{failure} & \text{if } i \geq 2^{31} \\ \text{HMAC-SHA256}(c_{par}, \text{ser}_p(K_{par}) \mid\mid \text{ser}_{32}(i)) & \text{if } i < 2^{31} \end{cases} \\
K_i &= l \cdot K_{par} \\
&= l \cdot (k_{par} \cdot G) \\
&= (l \cdot k_{par} \bmod n) \cdot G
\end{aligned}
$$

4.3.3 Child key share derivation

It is asume that one of the participants P_j is designated as the leader L. The function CKSD compute a child extended key share from the parent extended key share.

$$
\begin{aligned}
f(T) &= \begin{cases} \text{CKDpriv}(i) & \text{if } L = j \\ \text{CKDpub}(i) & \text{if } L \neq j \end{cases} \\
sk_i &= \left[\prod_{s=1}^{j} sk_{par}^{s} \right] \cdot T \\
&= sk_{par} \cdot T \\
pk_i &= pk_{par} \cdot T \\
&= sk_{par} \cdot G \cdot T \\
&= \left[\prod_{s=1}^{j} sk_{par}^{s} \right] \cdot G \\
c_i &= \text{HMAC-SHA256}(c_{par}, \text{ser}_{32}(i))
\end{aligned}
$$

If the index is greather or equal than 2^{31} the public key share of the participants $P_j = L$ need to be revealed in order to compute the public child key, a round of communication is needed.

4.4 Threshold deterministic signature

How to use an HMAC function in threshold mode to have deterministic k for signature.

5 Implementation in Bitcoin-core secp256k1

As mentionned before, Bitcoin use eliptic curve cryptography (ECC) for signing transactions. When the first release of Bitcoin core appeared in the early 2009, the cryptographic computations was performed with the OpenSSL library. Some years after a project started with the goal of replacing OpenSSL and creating a custom and minimalistic `C` library for cryptography over the curve secp256k1. This library is now available on GitHub at `bitcoin-core/secp256k1` project and it is one of the most optimized, if not the most optimized, library for the curve secp256k1. It is worth noting that this library is also used by other major crypto-currencies like Ethereum, so extending the capabilities of this library is a good choice to attract other cryptographer to have a look and increase the amount of reviews for this thesis.

The implementation is spread into four main components: (i) a `DER` parser-serializer, (ii) a textbook implementation of Paillier homomorphic cryptosystem, (iii) an implementation of the Zero-Knowledge Proofs adaptation, and (iv) the threshold public API. It is worth noting that the current implementation is NOT production ready and NOT side-channel attack resistant. Paillier and ZKP are not constant time computation and use `libgmp` for all arithmetic computations, even when secret values are used. This implementation is a textbook implementation of the scheme and need to be reviewed and more tested before been used in production. It is also worth noting that this library doesn't implement the functions needed to initialize the setup. Only the functions needed to parse existing keys and compute a distributed signature are implemented.

This chapter refers to the implementation available on GitHub at `https://github.com/GuggerJoel/secp256k1/tree/threshold` at the time when this lines are wrote. Note that the sources can evolve after that this report is written, to be sure to read the latest version of the code check out the sources directly on GitHub.

Contents

5.1 Configuration

The library use `autotools` to manage the compilation, installation and uninstallation. A system of module is already present in the structure with an ECDH experimental module for shared secret computation and a recovery module for recover ECDSA public key. A module can be flag as experimental, then, at the configuration time, an explicit parameter enabling experimental modules must be passed and a warning is shown to warn that the build contains experimental code.

5.1.1 Add new experimental module

In this structure, the threshold extension is all indicated to be an experimental module also. A new variable `$enable_module_recovery` is declared with a m4 macro defined by autoconf in the `configure.ac` file with the argument `--enable-module-threshold`. The default value is set to `no`.

```
137 AC_ARG_ENABLE(module_threshold,
138     AS_HELP_STRING([--enable-module-threshold],[enable Threshold ECDSA computation with
     ↪  Paillier homomorphic encryption system and zero-knowledge proofs (experimental)]),
139     [enable_module_threshold=$enableval],
140     [enable_module_threshold=no])
```

Listing 5.1   Add argument into `configure.ac` to enable the module

If the variable `$enable_module_recovery` is set to yes into `configure.ac` (lines 443 to 445) a compiler constant is declared, again with a m4 marco defined by autoconf, and set to 1 in `libsecp256k1-config.h` (lines 20 and 21.) This header file is generated when `./configure` script is run and is included in the library.

```
443 if test x"$enable_module_threshold" = x"yes"; then
444   AC_DEFINE(ENABLE_MODULE_THRESHOLD, 1, [Define this symbol to enable the threshold module])
445 fi
```

```
20 /* Define this symbol to enable the threshold module */
21 #define ENABLE_MODULE_THRESHOLD 1
```

Listing 5.2   Define constant `ENABLE_MODULE_THRESHOLD` if module enable

The main file `secp256k1.c` (lines 586 to 590) and the tests file `tests.c` include headers based on the compiler constant definition.

```
586 #ifdef ENABLE_MODULE_THRESHOLD
587 # include "modules/threshold/paillier_impl.h"
588 # include "modules/threshold/eczkp_impl.h"
589 # include "modules/threshold/threshold_impl.h"
590 #endif
```

Listing 5.3   Include implementation headers if `ENABLE_MODULE_THRESHOLD` is defined

The module is set to experimental to avoid enabling it without explicitly agree to build experimental code. If experimental is set to yes a warning is display during the configuration process, if experimental is not set and any experimental module is enable an error message is display and the process failed.

```
if test x"$enable_experimental" = x"yes"; then
  AC_MSG_NOTICE([******])
  AC_MSG_NOTICE([WARNING: experimental build])
  AC_MSG_NOTICE([Experimental features do not have stable APIs or properties, and may not be safe for production use.])
  AC_MSG_NOTICE([Building ECDH module: $enable_module_ecdh])
  AC_MSG_NOTICE([Building Threshold module: $enable_module_threshold])
  AC_MSG_NOTICE([******])
else
  if test x"$enable_module_ecdh" = x"yes"; then
    AC_MSG_ERROR([ECDH module is experimental. Use --enable-experimental to allow.])
  fi
  if test x"$enable_module_threshold" = x"yes"; then
    AC_MSG_ERROR([Threshold module is experimental. Use --enable-experimental to allow.])
  fi
  if test x"$set_asm" = x"arm"; then
    AC_MSG_ERROR([ARM assembly optimization is experimental. Use --enable-experimental to allow.])
  fi
fi
```

Listing 5.4 Set threshold module to experimental into `configure.ac`

5.1.2 Configure compilation

A module is composed of one or many `include/` headers that contain the public API with a small description of each functions, these headers are copied in the right folders when `sudo make install` command is run. The file `Makefile.am` define which headers need to be installed, which not and how to compile the project. This file is parsed by autoconf to generate the final `Makefile` with all the fonctionalities expected.

Each module has its own `Makefile.am.include` which describe what to do with all the files present into the module folder. This file is included in the main `Makefile.am` (lines 179 to 181) if the module is enable.

```
179 if ENABLE_MODULE_THRESHOLD
180 include src/modules/threshold/Makefile.am.include
181 endif
```

Listing 5.5 Include specialized Makefile if threshold module is enable

The specialized `Makefile.am.include` declare the header requisite to be include and declare the list of all the headers that must not be installed on the system when `sudo make install` command is run.

```
include_HEADERS += include/secp256k1_threshold.h
noinst_HEADERS += src/modules/threshold/der_impl.h
noinst_HEADERS += src/modules/threshold/paillier.h
noinst_HEADERS += src/modules/threshold/paillier_impl.h
noinst_HEADERS += src/modules/threshold/paillier_tests.h
noinst_HEADERS += src/modules/threshold/eczkp.h
noinst_HEADERS += src/modules/threshold/eczkp_impl.h
noinst_HEADERS += src/modules/threshold/eczkp_tests.h
noinst_HEADERS += src/modules/threshold/threshold_impl.h
noinst_HEADERS += src/modules/threshold/threshold_tests.h
```

Listing 5.6 Specialized Makefile for threshold module

It is possible to build the library and enable the threshold module with the command below.

```
./configure --enable-module-threshold --enable-experimental
```

5.2 DER parser-serializer

Transmit messages and retreive keys are an important part of the scheme. Because between all steps a communication on the network is necessary, a way to export and import data is required. Bitcoin private key are simple structures because of the fixed curve and their intrinsic nature, a single 2^{256} bits value. Threshold private key are composed of multiple parts like: (i) the private share, (ii) a Paillier private key, (iii) a Paillier public key, and (iv) Zero-Knowledge Proof parameters. To serialize these complex structures the `DER` standard has been choosed. Three simple data types are implemented in the library: (i) sequence, (ii) integer, and (iii) octet string.

5.2.1 Sequence

The sequence data structure holds a sequence of integers and/or octet strings. The sequence start with the constant `0x30` and is followed by the content lenght and the content itself. A lenght could be in the short form or the long form. If the content number of bytes is shorter to `0x80` the lenght byte indicate the lenght, if the content is equal or longer than `0x80` the seven lower bits 0 to 6 where $\texttt{byte} = \{\texttt{b}_7, \ldots, \texttt{b}_1, \texttt{b}_0\}$ indicate the number of followed bytes which are used for the lenght.

```c
void secp256k1_der_parse_len(const unsigned char *data, unsigned long *pos, unsigned long
    *lenght, unsigned long *offset) {
    unsigned long op, i;
    op = data[*pos] & 0x7F;
    if ((data[*pos] & 0x80) == 0x80) {
        for (i = 0; i < op; i++) {
            *lenght += data[*pos+1+i]<<8*(op-i-1);
        }
        *offset = op + 1;
    } else {
        *lenght = op;
        *offset = 1;
    }
    *pos += *offset;
}
```

Listing 5.7 Implementation of a `DER` lenght parser

The sequence parser check the first byte with the constant `0x30` and extract the content lenght. Position in the input array are holds in the `*pos` variable, extracted lenght is stored in `*lenght`, and the offset holds how many bytes in the data are used for the header and the lenght. A coherence check is performed to ensure that the current offset and the retreived lenght result to the same amount of bytes passed in argument.

When a sequence holds other sequence, retreive their total lenght (including header and content lenght bytes) is needed to recursivly parse them. A specific function is created to retreive the total lenght of a struct given a pointer to its first byte.

The serialization of a sequence is implemented as a serialization of an octet string with the sequence header `0x30` without integrity check of the content. The content lenght is serialized first, then the header is added.

```c
int secp256k1_der_parse_struct(const unsigned char *data, size_t datalen, unsigned long *pos, unsigned long *lenght, unsigned long *offset) {
    unsigned long loffset;
    if (data[*pos] == 0x30) {
        *pos += 1;
        secp256k1_der_parse_len(data, pos, lenght, &loffset);
        *offset = 1 + loffset;
        if (*lenght + *offset != datalen) { return 0; }
        else { return 1; }
    }
    return 0;
}
```

Listing 5.8 Implementation of a `DER` *sequence parser*

The result of a content lenght serialization can be ≥ 1 byte-s. If the content is shorter than `0x80`, then one byte is enough to store the lenght. Else multiple bytes (≥ 2) are used. Because the number of byte is undefined before the computation a memory allocation is necessary and a pointer is returned with the lenght of the array.

```c
unsigned char* secp256k1_der_serialize_sequence(size_t *outlen, const unsigned char *op, const size_t datalen) {
    unsigned char *data = NULL, *len = NULL;
    size_t lensize = 0;
    len = secp256k1_der_serialize_len(&lensize, datalen);
    *outlen = 1 + lensize + datalen;
    data = malloc(*outlen * sizeof(unsigned char));
    data[0] = 0x30;
    memcpy(&data[1], len, lensize);
    memcpy(&data[1 + lensize], op, datalen);
    free(len);
    return data;
}
```

Listing 5.9 Implementation of a `DER` *sequence serializer*

If the content lenght is longer than `0x80`, then `mpz` is used to serialize the lenght into a bytes array in big endian most significant byte first. The lenght of this serialization is stored into `longsize` and is used to create the first byte with the most significant bit set to 1 (line 93).

5.2.2 Integer

Integers are used to store the most values in the keys and Zero-Knowledge Proofs. An integer can be positive, negative or zero and are represented in the second complement form. The header start with `0x02`, followed by the lenght of the data. Parsing and serializing integer are already implemented in `libgmp`, functions are juste wrapper to extract information from the header and start the `mpz` importation at the right offset.

5.2.3 Octet string

Octet strings are used to holds serialized data like points/public keys. An octet string is an arbitrary array of bytes. The header start with `0x04` followed by the size of the content. The serialization implementation retreive the lenght of the content,

```
unsigned char* secp256k1_der_serialize_len(size_t *datalen, size_t lenght) {
    unsigned char *data = NULL; void *serialize; size_t longsize; mpz_t len;
    if (lenght >= 0x80) {
        mpz_init_set_ui(len, lenght);
        serialize = mpz_export(NULL, &longsize, 1, sizeof(unsigned char), 1, 0, len);
        mpz_clear(len);
        *datalen = longsize + 1;
    } else {
        *datalen = 1;
    }
    data = malloc(*datalen * sizeof(unsigned char));
    if (lenght >= 0x80) {
        data[0] = (uint8_t)longsize | 0x80;
        memcpy(&data[1], serialize, longsize);
        free(serialize);
    } else {
        data[0] = (uint8_t)lenght;
    }
    return data;
}
```

Listing 5.10 Implementation of a ***DER*** *lenght serializer*

copy the header and the octet string into a new memory space, and return the pointer with the total lenght. The parser implementation copy the content and set the conent lenght, the position index, and the offset.

5.3 Paillier cryptosystem

Homomorphic encryption is required in the scheme and Paillier is proposed in the white paper. Paillier homomorphic encryption is simple to implement in a textbook way, this implementation is functional but not optimized and need to be reviewed.

5.3.1 Data structures

Encrypted message, public and private keys are transmited. As mentionned before, the `DER` standard format is used to parse and serialize data. `DER` schema for all data structures are defined to ensure portability over different implementations.

Public keys

The public key is composed of a public modulus and a generator. The implementation data structure add a big modulus corresponding to the square of the modulus. A version number is added for future compatibility purposes.

```
HEPublicKey ::= SEQUENCE {
    version           INTEGER,
    modulus           INTEGER,  -- p * q
    generator         INTEGER
}
```

Listing 5.11 *`DER` schema of a Paillier public key*

`libgmp` is used for all the arithmetic in Paillier implementation, all numbers are stored in `mpz_t` type. The parser take in input an array of bytes with a lenght and the public key to fill.

```
typedef struct {
    mpz_t modulus;
    mpz_t generator;
    mpz_t bigModulus;
} secp256k1_paillier_pubkey;

int secp256k1_paillier_pubkey_parse(
    secp256k1_paillier_pubkey *pubkey,
    const unsigned char *input,
    size_t inputlen
);
```

Listing 5.12 *`DER` parser of a Paillier public key*

Private keys

The private key is composed of a public modulus, two primes, a generator, a private exponant $\lambda = \varphi(n) = (p-1)(q-1)$, and a private coefficient $\mu = \varphi(n)^{-1} \mod n$. Again, a version number is added for future compatibility purposes.

The parser take in input an array of bytes with a lenght and the private key to fill. The big modulus is computed after the parsing to accelerate encryption and decryption.

```
HEPrivateKey ::= SEQUENCE {
    version           INTEGER,
    modulus           INTEGER,   -- p * q
    prime1            INTEGER,   -- p
    prime2            INTEGER,   -- q
    generator         INTEGER,
    privateExponent   INTEGER,   -- (p - 1) * (q - 1)
    coefficient       INTEGER    -- (inverse of privateExponent) mod (p * q)
}
```

Listing 5.13 DER *schema of a Paillier private key*

```
typedef struct {
    mpz_t modulus;
    mpz_t prime1;
    mpz_t prime2;
    mpz_t generator;
    mpz_t bigModulus;
    mpz_t privateExponent;
    mpz_t coefficient;
} secp256k1_paillier_privkey;

int secp256k1_paillier_privkey_parse(
    secp256k1_paillier_privkey *privkey,
    secp256k1_paillier_pubkey *pubkey,
    const unsigned char *input,
    size_t inputlen
);
```

Listing 5.14 DER *parser of a Paillier private key*

Encrypted messages

An encrypted message with Paillier cryptosystem is a big number $c \in \mathbb{Z}^*_{n^2}$. No version number is added in this case. The implementation structure contain a nonce value that could be set to 0 to stores the nonce used during encryption.

```
HEEncryptedMessage ::= SEQUENCE {
    message           INTEGER
}
```

Listing 5.15 DER *schema of an encrypted message with Paillier cryptosystem*

An encrypted message can be serialized and parsed and they are used in messages exchange during the signing protocol by both parties.

5.3.2 Encrypt and decrypt

Like all other encryption schemes in public key cryptography, the public key is used to encrypt and the private key to decrypt. To encrypt the message `mpz_t m` where $m < n$, a random value r where $r < n$ is selected with the fonction pointer `noncefp` and set into the nonce value `res->nonce`. This nonce is stored because his value is needed to create Zero-Knowledge Proofs. Then, the cipher $c = g^m \cdot r^n \mod n^2$ is putted into `res->message` to complete the encryption process. All intermediray states are wipe out before returning the result.

```c
int secp256k1_paillier_encrypt_mpz(secp256k1_paillier_encrypted_message *res, const mpz_t m,
 ↪  const secp256k1_paillier_pubkey *pubkey, const secp256k1_paillier_nonce_function
 ↪  noncefp) {
    mpz_t l1, l2, l3;
    int ret = noncefp(res->nonce, pubkey->modulus);
    if (ret) {
        mpz_inits(l1, l2, l3, NULL);
        mpz_powm(l1, pubkey->generator, m, pubkey->bigModulus);
        mpz_powm(l2, res->nonce, pubkey->modulus, pubkey->bigModulus);
        mpz_mul(l3, l1, l2);
        mpz_mod(res->message, l3, pubkey->bigModulus);
        mpz_clears(l1, l2, l3, NULL);
    }
    return ret;
}
```

Listing 5.16 Implementation of encryption with Paillier cryptosystem

If the random value selection process failed the encryption fail also. The random function of type `secp256k1_paillier_nonce_function` must use a good CPRNG and his implementation is not part of the library.

```c
typedef int (*secp256k1_paillier_nonce_function)(
    mpz_t nonce,
    const mpz_t max
);
```

Listing 5.17 Function signature for Paillier nonces generation

To decrypt the cipher $c \in \mathbb{Z}^*_{n^2}$ with the private key, the function compute $m = L(c^\lambda \mod n^2) \cdot \mu \mod n$ where $L(x) = (x-1)/n$. The cipher is raised to the lambda $c^\lambda \mod n^2$ in line 4 and the result is putted to an intermediray state variable. Then the $L(x)$ function is applied on the intermediray state in lines 5-6. Finally, the multiplication with μ and the modulo of n are taken (lines 7-8) to lead to the result. It is worth noting that, in line 6, only the quotient of the division is recovered.

```c
1  void secp256k1_paillier_decrypt(mpz_t res, const secp256k1_paillier_encrypted_message *c,
    ↪  const secp256k1_paillier_privkey *privkey) {
2      mpz_t l1, l2;
3      mpz_inits(l1, l2, NULL);
4      mpz_powm(l1, c->message, privkey->privateExponent, privkey->bigModulus);
5      mpz_sub_ui(l2, l1, 1);
6      mpz_cdiv_q(l1, l2, privkey->modulus);
7      mpz_mul(l2, l1, privkey->coefficient);
8      mpz_mod(res, l2, privkey->modulus);
9      mpz_clears(l1, l2, NULL);
10 }
```

Listing 5.18 Implementation of decryption with Paillier cryptosystem

5.3.3 Homomorphism

The choice of this scheme is not hazardous, homomorphic addition and multiplication are used to construct the signature composant $s = D_{sk}(\mu) \mod q : \mu = (\alpha \times_{pk} m'z_2) +_{pk} (\zeta \times_{pk} r'x_2z_2) +_{pk} E_{pk}(cn)$ where $+_{pk}$ denotes homomorphic addition over the ciphertexts and $\times_{pk}$ denotes homomorphic multiplication over the ciphertexts.

Addition

Addition $+_{pk}$ over ciphertexts is computed with $D_{sk}(E_{pk}(m_1, r_1) \cdot E_{pk}(m_2, r_2) \mod n^2) = m_1 + m_2 \mod n$ or $D_{sk}(E_{pk}(m_1, r_1) \cdot g^{m_2} \mod n^2) = m_1 + m_2 \mod n$ where D_{sk} denotes descryption with private key sk and E_{pk} denotes encryption with public key pk. Only the first variant is implemented, where two ciphertexts are added together to result in a third ciphertext.

```
void secp256k1_paillier_add(secp256k1_paillier_encrypted_message *res, const
↪   secp256k1_paillier_encrypted_message *op1, const secp256k1_paillier_encrypted_message
↪   *op2, const secp256k1_paillier_pubkey *pubkey) {
    mpz_t l1;
    mpz_init(l1);
    mpz_mul(l1, op1->message, op2->message);
    mpz_mod(res->message, l1, pubkey->bigModulus);
    mpz_clear(l1);
}
```

Listing 5.19 Implementation of homomorphic addition with Paillier cryptosystem

Multiplication

Multiplication $\times_{pk}$ over ciphertexts can be performed with $D_{sk}(E_{pk}(m_1, r_1)^{m_2} \mod n^2) = m_1 m_2 \mod n$, the implementation is straight forward in this case. The nonce value from the ciphertext is copied in the resulted encrypted message for not lose information after opperations.

```
void secp256k1_paillier_mult(secp256k1_paillier_encrypted_message *res, const
↪   secp256k1_paillier_encrypted_message *c, const mpz_t s, const secp256k1_paillier_pubkey
↪   *pubkey) {
    mpz_powm(res->message, c->message, s, pubkey->bigModulus);
    mpz_set(res->nonce, c->nonce);
}
```

Listing 5.20 Implementation of homomorphic multiplication with Paillier cryptosystem

5.4 Zero-knowledge proofs

Two Zero-Knowledge Proofs are used in the scheme, each party generate a proof and validates the other one. A proof is generated and verified under some ZKP parameters, these parameters are fixed at the initialization time and don't change over the time.

5.4.1 Data structures

Three data structures are created, one for each ZKP and one for storing the parameters. Zero-Knowledge Proofs are composed of big numbers and points and need to be serialized and parsed to be included in the messages exchange protocol.

Zero-Knowledge Parameters

Zero-Knowledge parameter is composed of three numeric values: (i) $\tilde{N}$ a public modulus, (ii) h_2 a value selected randomly $\in \mathbb{Z}^*_{\tilde{N}}$, and (iii) h_1 a value where $\exists x, \log_x(h_1) = h_2 \mod \tilde{N}$. One function is provided in the module to parse a `ZKPParameter` `DER` schema.

```
ZKPParameter ::= SEQUENCE {
    modulus           INTEGER,
    h1                INTEGER,
    h2                INTEGER
}
```

Listing 5.21 *`DER` schema of a Zero-Knowledge parameters sequence*

Zero-Knowledge Proof Π

Zero-Knowledge Proof Π is composed of numeric values and one point. The point is stored in a public key internal structure inside the implementation and is exported with the secp256k1 library as a 65 bytes uncompressed public key. The uncompressed public key is then stored as an octet string in the schema. A version number is added for future compatibility purposes. Two functions are provided in the module to parse and serialize a `ECZKPPi` `DER` schema.

```
ECZKPPi ::= SEQUENCE {
    version           INTEGER,
    z1                INTEGER,
    z2                INTEGER,
    y                 OCTET STRING,
    e                 INTEGER,
    s1                INTEGER,
    s2                INTEGER,
    s3                INTEGER,
    t1                INTEGER,
    t2                INTEGER,
    t3                INTEGER,
    t4                INTEGER
}
```

Listing 5.22 *`DER` schema of a Zero-Knowledge Π sequence*

Zero-Knowledge Proof Π'

Zero-Knowledge Proof Π' is composed of the same named values as ZKP Π plus five new ones. The construction of the proof is based on Π but needs more than values to express all the proven statements. Again, the point y is a point serialized as an uncompressed public key in an octet string and a version number is added for future compatibility purposes. Two functions are provided in the module to parse and serialize a `ECZKPPiPrim` `DER` schema.

```
ECZKPPiPrim ::= SEQUENCE {
    version           INTEGER,
    z1                INTEGER,
    z2                INTEGER,
    z3                INTEGER,
    y                 OCTET STRING,
    e                 INTEGER,
    s1                INTEGER,
    s2                INTEGER,
    s3                INTEGER,
    s4                INTEGER,
    t1                INTEGER,
    t2                INTEGER,
    t3                INTEGER,
    t4                INTEGER,
    t5                INTEGER,
    t6                INTEGER,
    t7                INTEGER
}
```

Listing 5.23 *DER schema of a Zero-Knowledge Π' sequence*

5.4.2 Generate proofs

Proofs are generated in relation to a specific setup and a specific in progress signature. which makes them linked to a large number of values (points, encrypted messages, secrets, parameters, etc.) The complexity of these constructions is strongly felt in the code. Heavy mathematic computations are needed with two `hash` functions.

A CPRNG function is required to generate both proofs. This function generate random number in $\mathbb{Z}_{max}$ and $\mathbb{Z}^*_{max}$. The `flag` argument indicate which case is treated, `STD` or `INV`. If the function have not access to a good source of randomness or cannot generate a good random number a zero is returned, otherwise a one is returned.

```
typedef int (*secp256k1_eczkp_rdn_function)(
    mpz_t res,
    const mpz_t max,
    const int flag
);

#define SECP256K1_THRESHOLD_RND_INV 0x01
#define SECP256K1_THRESHOLD_RND_STD 0x00
```

Listing 5.24 *Function signature for ZKP CPRNG*

Zero-Knowledge Proof Π

As shown in FIGURE REF, the proof states that: (i) it exists a known value by the proover that link $r \to r_2$, (ii) it exists a second known value by the proover that, related to the first one, link $G \to y_1$, (iii) the result of $D_{sk}(\alpha)$ is this first value, and (iv) the result of $D_{sk}(\zeta)$ is this second value.

To do computation on the curve a context object need to be passed in argument, then the ZKP object to fill, the ZKP parameters, the two encrypted messages α and ζ, scalar values sx_1 and sx_2 representing $z_1 = (k_1)^{-1} \mod n$ and x_1z_1, then the point r, the point r_2, the partial public key y_1, the proover Paillier public key which has been used to encrypt α and ζ, and finally a pointer to a CPRNG function used to generate all needed random values.

```
int secp256k1_eczkp_pi_generate(
    const secp256k1_context *ctx,
    secp256k1_eczkp_pi *pi,
    const secp256k1_eczkp_parameter *zkp,
    const secp256k1_paillier_encrypted_message *m1,
    const secp256k1_paillier_encrypted_message *m2,
    const secp256k1_scalar *sx1,
    const secp256k1_scalar *sx2,
    const secp256k1_pubkey *c,
    const secp256k1_pubkey *w1,
    const secp256k1_pubkey *w2,
    const secp256k1_paillier_pubkey *pubkey,
    const secp256k1_eczkp_rdn_function rdnfp
);
```

Listing 5.25 Function signature to generate ZKP Π

The function implementation can be splited in four main parts: (i) generate all the needed random values v, (ii) compute the challenge values, (iii) compute the `hash` of these values v, and (iv) compute the ZKP values with $e = \texttt{hash}(v)$.

Zero-Knowledge Proof Π'

As shown in FIGURE REF, the proof states that: (i) it exists a known value by the proover x_1 that link $r_2 \rightarrow G$, (ii) it exists a second known value by the proover that, related to the first one, link $G \rightarrow y_2$, (iii) the result of $D_{sk'}(\mu')$ is this first value, and (iv) it exists a third known value by the proover x_3 and the result of $D_{sk}(\mu)$ is the homomorphic operation of $(\alpha \times x_1) + (\zeta \times x_2) + x_3$.

```
int secp256k1_eczkp_pi2_generate(
    const secp256k1_context *ctx,
    secp256k1_eczkp_pi2 *pi2,
    const secp256k1_eczkp_parameter *zkp,
    const secp256k1_paillier_encrypted_message *m1,
    const secp256k1_paillier_encrypted_message *m2,
    const secp256k1_paillier_encrypted_message *m3,
    const secp256k1_paillier_encrypted_message *m4,
    const secp256k1_paillier_encrypted_message *r,
    const mpz_t x1,
    const mpz_t x2,
    const mpz_t x3,
    const mpz_t x4,
    const mpz_t x5,
    const secp256k1_pubkey *c,
    const secp256k1_pubkey *w2,
    const secp256k1_paillier_pubkey *pairedkey,
    const secp256k1_paillier_pubkey *pubkey,
    const secp256k1_eczkp_rdn_function rdnfp
);
```

Listing 5.26 Function signature to generate ZKP Π'

The function implementation can also be splited in four main parts: (i) generate all the needed random values v, (ii) compute the proof values, (iii) compute the `hash'` of these values v, and (iv) compute the ZKP values with $e = \texttt{hash'}(v)$.

It is worth noting that `hash` and `hash'` must be different hashing function to avoid reusing Π proofs, even not satisfying the predicate, to construct fraudulent Π' proofs.

5.4.3 Validate proofs

Validation of proofs Π and Π' can be done with: (i) the Paillier public keys, (ii) the ZKP parameters, and (iii) the exchanged messages. The process can be splited in three steps: compute the proof values, retreive the candidate value e', and compare if $e = e'$. If the values match the proof is valid.

```
int secp256k1_eczkp_pi_verify(
    const secp256k1_context *ctx,
    secp256k1_eczkp_pi *pi,
    const secp256k1_eczkp_parameter *zkp,
    const secp256k1_paillier_encrypted_message *m1,
    const secp256k1_paillier_encrypted_message *m2,
    const secp256k1_pubkey *c,
    const secp256k1_pubkey *w1,
    const secp256k1_pubkey *w2,
    const secp256k1_paillier_pubkey *pubkey
);

int secp256k1_eczkp_pi2_verify(
    const secp256k1_context *ctx,
    secp256k1_eczkp_pi2 *pi2,
    const secp256k1_eczkp_parameter *zkp,
    const secp256k1_paillier_encrypted_message *m1,
    const secp256k1_paillier_encrypted_message *m2,
    const secp256k1_paillier_encrypted_message *m3,
    const secp256k1_paillier_encrypted_message *m4,
    const secp256k1_pubkey *c,
    const secp256k1_pubkey *w2,
    const secp256k1_paillier_pubkey *pubkey,
    const secp256k1_paillier_pubkey *pairedkey
);
```

Listing 5.27 *Function signature to validate ZKP* Π *and* Π'

5.5 Threshold module

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

6 Further research

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

6.1 Hardware wallets

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

7 Conclusions

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

A Docker Configuration

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

```yaml
version: '2.1'
services:
  levee:
    command: >
        bash -c "echo Container ready! Sleep 100000000... && sleep 100000000"
    privileged: true
    container_name: levee
    build: ./levee
    image: levee/levee
    volumes:
        - ./levee/shared:/shared
```

Listing A.1 *caption*

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

List of Figures

List of Tables

List of sources

Bibliography

[1] Philip D. MacKenzie and Michael K. Reiter. "Two-Party Generation of DSA Signatures". In: *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*. Vol. 2139. Lecture Notes in Computer Science. Springer, 2001, pp. 137–154. DOI: `10.1007/3-540-44647-8_8`.

[2] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. "Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security". In: *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings*. 2016, pp. 156–174. DOI: `10.1007/978-3-319-39555-5_9`. URL: `https://doi.org/10.1007/978-3-319-39555-5_9`.

[3] Pascal Paillier. "Public-key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT'99. Prague, Czech Republic: Springer-Verlag, 1999, pp. 223–238. ISBN: 3-540-65889-0. URL: `http://dl.acm.org/citation.cfm?id=1756123.1756146`.

[4] Steven Goldfeder et al. "Securing Bitcoin wallets via a new DSA/ECDSA threshold signature scheme". In: 2015.

[5] Mihir Bellare and Phillip Rogaway. "Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. CCS '93. Fairfax, Virginia, USA: ACM, 1993, pp. 62–73. ISBN: 0-89791-629-8. DOI: `10.1145/168588.168596`. URL: `http://doi.acm.org/10.1145/168588.168596`.