
Performance Analysis of an Offline Digital Euro Prototype

Robbert Koning

To obtain the degree of Master of Science in Computer Science
Data Science & Technology Track
To be defended publicly on March 31, 2023

Student number: 4588576
Thesis committee: Dr. ir. J.A. Pouwelse, supervisor
Dr. G. Lan

Delft University of Technology
Faculty of Electrical Engineering, Mathematics & Computer Science
Distributed Systems Group

Performance Analysis of an Offline Digital Euro Prototype

Robbert Koning, Johan Pouwelse (thesis supervisor)

Distributed Systems
Delft University of Technology
Delft, The Netherlands

—master’s thesis—

Abstract—Recent years have seen an increasing interest in stablecoins from major corporate and governmental parties. The European Central Bank is investigating the possibility of introducing its own Central Bank Digital Currency. The desired features of such a currency are under discussion. One such feature is offline spending: the ability to use the currency without an internet connection, like cash. This thesis describes a token-based transaction prototype and its implementation on the Kotlin-IPv8 protocol stack. The prototype allows funds to be spent in an offline setting and provides retroactive fraud detection. The prototype is not intended for deployment but instead serves as a trial for building digital currencies on Kotlin-IPv8. The included performance analysis demonstrates that various facets of Kotlin-IPv8 perform suboptimally, of which most notably its UDP data throughput.

I. INTRODUCTION

In recent years, the European Central Bank (ECB) has been exploring the possibility of realizing its own Central Bank Digital Currency (CBDC), the ‘digital Euro’. The ECB has published various reports and resources that outline the need for such a project (i.e. [1]–[3]). Calls for expression of interest are being published and the ECB aims to complete its investigation phase by October 2023 [4], [5]. The main reason for this development is the rise of digital payments and corresponding decline of cash usage. According to reports published by De Nederlandsche Bank (DNB), the national bank of the Netherlands, the share of cash payments dropped from 56% in 2010 to 21% in 2020 [6], [7]. The Swedish Riksbank mentions similar trends for Sweden [8].

Euro cash is the only public form of money that is directly backed by the ECB [2]. Digital payments are not; they are backed by private parties such as commercial banks. A critical dependence on these parties can erode the sovereignty of the Euro. They cannot safeguard reliability comparable to that of ECB-backed cash. Nevertheless, there is demand for reliability, especially in times of crisis [9]. In recent history, there have been several financial crises that caused large-scale bankruptcies which consequently impacted consumers’ savings (e.g. in 2008). CBDCs can provide reliability and safeguard consumers against the effects of large-scale bankruptcy of commercial payment providers.

Foreign organisations, commercial parties, and cryptocurrencies are threatening the influence of central banks. A report published by the ECB discusses the risk of *currency substitution*. Substitution occurs when a new form of money, unregulated by the ECB, gains major usage in the EU. The new payment method would likely have to outperform its competitors, for instance by being cheaper and/or more convenient. According to the report, currency substitution could have a range of negative effects on the ECB’s monetary policy and even threaten the EU’s independence [1]. Along with the ECB, other major governmental and commercial parties have also shown interest in developing a CBDC. Some interested governmental parties are e.g. the United States government and the People’s Bank of China [10], [11]. An interested commercial party is for instance Meta (formerly Facebook), which initiated *Diem*¹, a hypothesized stablecoin that did not launch due to legal and regulatory issues.

Due to the potentially far-reaching impact of the introduction of CBDCs to consumers and the rapid pace with which central banks are operating, public discourse on the topic has been increasing. Some regulators are questioning the use case of CBDCs and their implications on the role of central banks and consumers’ privacy [12], [13].

To determine the appropriate role for a CBDC in the EU, and to compete with other payment solutions, the ECB has launched an extensive exploratory phase. In this exploration, the ECB has expressed interest for its CBDC to be usable in an offline environment [1]. This is crucial in case of network failure or in areas without a reliable internet connection. A prominent example of currency that is spent offline is cash.

This thesis concerns itself with implementing a simple transferable digital currency on the Kotlin-IPv8² protocol stack and doing a performance analysis. In accordance with the *Offline First* design principles, the currency can be spent offline and guarantees retroactive fraud detection. It is therefore resilient against temporary failure of central servers, unlike many currently deployed systems. This thesis contributes 1) a software-implemented simple token-based transaction system

¹For Diem, refer to <https://www.diem.com/en-us/>.

²For Kotlin-IPv8 and EVA, refer to <https://github.com/Tribler/kotlin-ipv8>.

2) a performance analysis of various bottlenecks in this system, with a special emphasis on the Kotlin-IPv8 framework and 3) a slightly optimized version of the EVA² protocol [14].

II. PROBLEM DESCRIPTION

The main difficulty with implementing offline digital currencies is solving or mitigating the *double spending problem*. Double spending is the action of spending a digital unit of value more than once, illegitimately. In a digital environment, data is easily duplicated. Thus, digital currencies must be made resistant against duplication of data. When transacting parties can only communicate to those within their physical proximity, meaning they are *offline*, fraud prevention becomes difficult. This is because they cannot consult with others about their counterparty. For example, common financial systems rely on banks to verify that counterparties possess enough funds to make a transaction. When offline, parties cannot contact their bank, and thus cannot confirm the validity of their transactions. The double spending problem has thus far not been solved in an offline setting, only in an online setting.

Many cryptocurrencies (e.g. Bitcoin) mitigate the problem by utilizing *global consensus* [15]. This removes the need for a central authority but does require near-immediate connectivity to parts of the network. Global consensus disallows offline transfers and is therefore not the right solution to make offline spending possible.

Since the first proposed digital currency in 1983 (see Section III-A), the problem of offline spending has been explored extensively. Many offline currency schemes in the field are *token-based*, as opposed to *account-based*. Token-based schemes transfer tokens: monetary units that can be identified with a serial number. By contrast, account-based schemes perform monetary transfers by crediting and debiting accounts. The crucial difference is that currency in token-based schemes is identifiable, whereas in account-based schemes it is not. A commonly used analogy is that token-based schemes are comparable to banknotes, whereas account-based schemes are comparable to bank deposits. A crucial lesson observed from the literature and our main prior work (see Section III-B), is that account-based systems complicate robustness measures such as safeguarding against double spending [16].

Another major problem is that realizing a proposed digital currency is a difficult engineering challenge. For instance, scalability and security need not only be accounted for in the design of a system but also in its implementation.

III. RELATED WORK

A. Advancements in digital currency

In 1983, Chaum introduced blind signatures in what is widely accredited as the first paper to describe digital currency [17]. The paper describes a novel cryptographic primitive, the *blind signature*. It allows parties to sign messages without knowing their contents. The result is that the signing party cannot relate their own signature to the original message they signed. With this primitive, the literature’s first digital cash scheme was described. In this scheme, a monetary authority

guarantees the validity of payments. Due to blind signatures, the authority cannot identify the recipient of any transaction it verifies, thereby safeguarding consumers’ privacy. Chaum’s cash was however *non-transferable*. Non-transferable e-cash can be spent only once, after which it must be redeemed by a trusted authority. The authority returns an equivalent amount of cash that is spendable again.

In 1989, Okamoto introduced *transferable* e-cash [18]. Transferable e-cash is more like physical cash; it can be spent repeatedly from one user to another. It does not require a network connection to a monetary authority with every transaction. In the same paper, *divisible* e-cash was introduced. In contrast to physical cash, divisible e-cash can be spent in smaller denominations than the piece that is owned. An advantage of divisible e-cash is that exact payments can be made and change is not required.

In 1995, a modification to blind signatures was proposed that made them ‘fair’ [19]. Most blind signature schemes were *perfectly unlinkable*. Perfect unlinkability means that no monetary authority can relate withdrawals to payments. Therefore, these schemes allowed for a variety of crimes to be undetectable, such as money laundering. With the introduction of ‘fair’ blind signatures, an additional and independent authority (such as a judge) would be able to obtain information that can be used to detect crime.

In 2008, Bitcoin was presented, widely accredited as the first major cryptocurrency. It solves the double spending problem probabilistically and without a central authority [15]. Bitcoin’s value is determined by market forces and is highly volatile. This is in stark contrast to CBDCs, which are tethered in value to government-issued money.

B. Eurotoken

We consider the main prior work for this thesis to be the first Eurotoken prototype by Delft University of Technology [20]. This digital currency is also implemented on Kotlin-IPv8. Eurotoken is an account-based system and is non-transferable by default. Eurotoken opted for a trusted authority to verify transactions. It is therefore governed in a centralized manner. The advantage of this approach in the context of CBDCs is that it enables the respective central bank to exert control over the network. Moreover, it provides a non-deterministic near-immediate transaction finality.

Based upon Eurotoken and in line with many proposed digital cash schemes, we also sacrificed governmental decentralization and opted for a centralized monetary authority. By contrast, our prototype is token-based instead of account-based and offline transferable by design.

C. Price Stability

It is fundamental for a European CBDC to be tethered in value to the Euro. A high price volatility like Bitcoin’s is undesirable for a medium of exchange [21]. There are various ways in which the value of an asset can be kept stable. This topic has gained renewed interest with the rise of *stablecoins*—cryptocurrencies that aim to be non-volatile

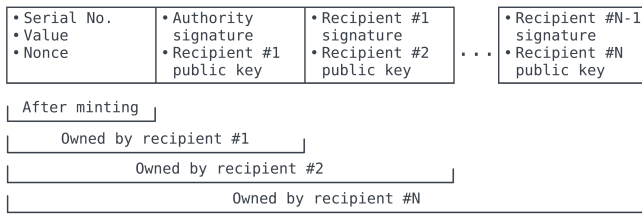


Fig. 1: Graphical representation of a token. Tokens represent monetary units of fixed value that store all their previous recipients until they are verified by an authority.

with regards to a major non-cryptocurrency or physical asset. There is an inverse relationship between the potential stability of stablecoins and how much they are decentralized [22]. The strongest stabilization mechanism is collateralization by currency or off-chain assets such as gold. By allowing free trade between a stablecoin and its collateral at a fixed price, arbitrage prevents the stablecoin’s price from fluctuating greatly. However, off-chain assets are not traded in a decentralized way and as such there is a trade-off between decentralization and stability. To the best of our knowledge, no decentralized and highly stable stablecoins exist. The prototype described in this thesis does not have a functional exchange to keep it stable with regards to the Euro. The implementation of this or other means of maintaining price stability is intentionally left out of scope.

IV. DESIGN AND ARCHITECTURE

This thesis implements a centralized CBDC prototype that allows offline transactions with fixed-value tokens and guarantees retroactive fraud detection.

The proposed system requires a trusted monetary authority that is in charge of token exchange and transaction verification. We refer to this party as *authority* and identify them by their public key. Verification is therefore a centralized operation. The motivation for this design choice was elaborated upon in Section III-B.

All system participants apart from the authority are clients. They, too, are identified by their public key. It is assumed that clients know the public key of the authority in the network. Clients can transact tokens to each other and consult the authority to verify the validity of their tokens. If clients cannot connect to the authority, for instance during a power outage, they can continue transacting but defer verification until they can connect.

To realize retroactive fraud detection, the implemented system requires authorities to be able to unambiguously reconstruct the sequence of owners of a token. This is done by providing each token with a linked list of all previous owners until its last verification. Details of this procedure are explained further in this section.

A. Token Format

The token protocol is based upon transacting tokens. A diagram of a token is given in Figure 1. Each token contains:

- 1) *Serial number*. An 8-byte unique token identifier.
- 2) *Value*. A 1-byte representation of the token’s worth. Like cash, tokens have a limited number of fixed denominations; the remaining values are considered invalid.
- 3) *Authority public key*. A 74-byte public key³ of the authority that is in charge of the token (the ‘authority’).
- 4) *Nonce*. A 64-byte pseudo-random nonce used by the authority to differentiate between differing occasions where the same token is sent to the same recipient.
- 5) *Recipients*. A list of recipient-proof pairs in chronological order. This list must contain at least a first pair:
 - a) *First recipient public key*. A 74-byte public key³ of the token’s first recipient after creation or validation.
 - b) *First proof*. A 64-byte signature (‘proof’) given by the authority signing *serial number*, *value*, *nonce*, and *first recipient public key*.

All pairs in the list are of the same format and bit-length. The second pair (if present) contains *second recipient public key* and a signature given by *first recipient public key* signing *first proof* and *second recipient public key* together. Likewise, all subsequent pairs follow the same pattern; they contain a signature from the previous public key in the list, signing the previous proof together with the next public key. This signature chain corresponds to the token changing ownership during transactions.

- 6) *Number of recipients*. A 2-byte number counting how many recipients are in *recipients*. This number is used to (de)serialize individual tokens from data files.

The initial size of a token when transferred from a monetary authority to the first recipient adds up to 287 bytes. Each additional recipient adds $74 + 64 = 138$ bytes for its public key and signature, respectively. Thus, for $k > 0$ recipients, the size of a token in bytes is defined as:

$$size = 287 + (k - 1) \cdot 138 \quad (1)$$

When a token is transferred to the first recipient, 147 of its bytes need to be cryptographically verified. The 140 bytes that do not need to be verified consist of *authority public key* (74 bytes), *first proof* (64 bytes), and *number of recipients* (2 bytes). Each additional recipient adds 138 bytes that need to be verified; 74 for its own public key and 64 for the previous proof in the list. For $k > 0$ recipients, k independent cryptographic signatures need to be verified, amounting to a total number of bytes defined as:

$$bytes = 147 + (k - 1) \cdot 138 \quad (2)$$

When a token is verified by a monetary authority, its size is reset to 287 bytes (see Section IV-E). The bit-lengths of the signatures and public keys were adapted from those used in Kotlin-IPv8 and are not integral to the protocol’s functioning.

³Public keys in Kotlin-IPv8 are 74 bytes long: 10 bytes for a string prefix; 32 bytes for an encryption key; and 32 bytes for a verification key. Only the latter is required for our prototype. However, Kotlin-IPv8 does not allow these to be split by design, as parties are identified by the entire 74 bytes.

B. Token Minting

When a token is created, its *serial number*, *value*, *nonce*, *authority public key*, and *recipients* are set as specified in Section IV-A. The authority stores a copy of the entire token and sends it to the intended client.

C. Client Verification

When a client obtains a token, it verifies it in a 3-step process. First, the client verifies that the token's last recipient (that is, the last public key in *recipients*) refers to them. Second, the client verifies that it knows the token's *authority public key* and that this key created the token's *first proof*. Third, the client verifies the remaining chain of proofs in *recipients*. The purpose of the client's verification process is merely to ensure that they have received an unambiguous proof of transfer from their transaction's counterparty. This proof can later be used by the relevant authority to prove potential fraud. A client deciding that a token is valid does not imply that an authority will decide the same. The client's verification does however guarantee that clients victimized by fraud can prove so eventually.

D. Client Transaction

A token's initial recipient may choose to send it to another client. If it does, it must append a new pair to the token's *recipients* that contains the desired recipient's public key and a signature combining the token's last proof together with the desired recipient's public key. This is depicted in Figure 1.

If the other client receives the token, it will verify it as specified in Section IV-C. A client that did not receive their token from the authority directly may opt to send it to their authority for verification. Sending a token to the authority is done in the same manner as sending it to a client, except that the public key of the recipient must be that of the authority. However, clients may also be obstructed from sending their token to the authority. Transactions can still proceed in situations without a connection to the authority. However, it may be that unverified tokens have been double spent.

E. Authority Verification

The authority's verification process is started when a client sends them a token to verify. The verification process consists of 6 steps:

- 1) The authority ensures that the incoming token has at least 3 recipients in its *recipients* list. There should be at least an initial recipient, a second recipient, and the authority itself as the last recipient. If not, the token is either invalid or ineligible for verification. Tokens can contain more than 3 recipients. In such cases, the token was spent offline. In all cases, the authority must be the last recipient in the *recipients* list.
- 2) The authority ensures that the token's penultimate recipient is the client that sent the token in for verification.
- 3) The authority queries if the token is still valid. The knowledge that the authority once signed the received

token, which can be derived from the token's *first proof*, says little about the token's current state. The authority compares its public key against the token's *authority public key* and queries the token's *serial number* to ensure that they are the authority that manages the token. Then it verifies that the token is still in circulation and not e.g. blacklisted.

- 4) The authority will, like an honest client, verify the chain of proofs in the *recipients* list.
- 5) The authority will attempt to detect double spending. It will compare the *first proof* of the incoming token to the last proof of its own copy of the token. If these are identical, double spending might have occurred but cannot be proven (see Section IV-F) and the authority will finalize verification. Finalizing verification requires the authority to update its own copy of the token by appending all new recipient-proof pairs of the received token to its *recipients*. The result is that the authority owns a copy of the token that contains its entire verified history, from its initial minting to its last known recipient.
- 6) The authority copies the incoming token, empties the *recipients* list save for its last entry, and sends the verified token to the desired recipient. It will also append the desired recipient to its own *recipients*.

F. Double Spending Detection

In Section IV-E it is mentioned that the authority updates the *recipients* of its own copy of a token upon a valid verification. To detect double spending, an authority compares the last proof of its own copy to *first proof* of an incoming token. A diagram of this scenario is depicted in Figure 2. If a token is double spent then multiple versions of the token will eventually reach their authority. The first time a token reaches their authority, double spending cannot be detected and the token-copy is updated. Subsequent times, the authority's token-copy already has an updated *recipients* list and therefore its last proof will not correspond to the double spent token's *first proof* anymore. Thus, double spending must have occurred if the proofs differ. If the proofs are equal, double spending might have occurred but cannot be proven. If a double spent token reaches the authority earlier than its legally-spent counterpart, the illicit transaction will only be detected when a conflicting transaction comes in.

When double spending is detected, the authority will search for the instigator. It will find the received token's *first proof* in the *recipients* of its token-copy. It will then compare the recipient-proof pairs of the token-copy with those of the received token. Comparison starts from the pairs that contain *first proof*. All pairs before it have already been verified. Eventually, it must find two differing pairs, after which all pairs will be different because proofs are chained to each other. The first differing pairs are the start of the token's split history and prove that double spending was performed by the client that signed them. The repercussions of implicating a client with fraud are further discussed in Section IV-I

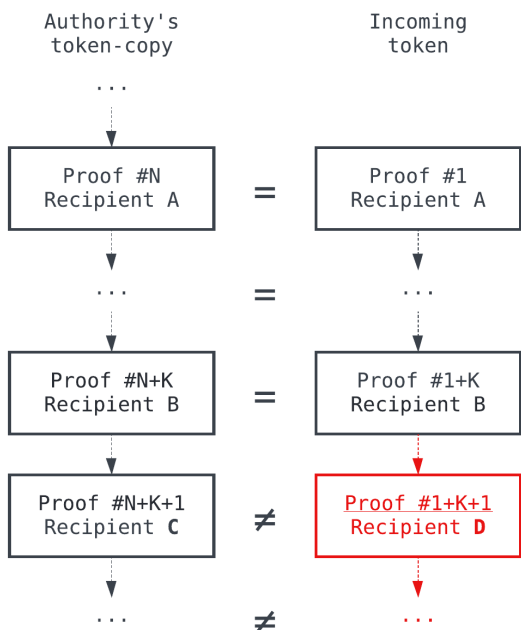


Fig. 2: The authority’s double spending detection mechanism. In the figure, a client sent their token to the authority for verification. The token was initially sent to the authority to client A and reached client B after K transactions. The authority compares its own version of the token (left) to the incoming token (right). It detects that B double spent a token, which was detected because proof $N + K + 1$ of the authority was not equal to proof $1 + K + 1$ of the incoming token.

G. Replay Attack Prevention

The detection mechanism of Section IV-F allows for a replay attack in an offline environment. If a malicious sender A were to replay sending the same token to the same receiver B as before, said receiver would not flag this as malicious behavior. If B in turn were to spend this token, upon verification of the token, B would be flagged as a double spender. When an authority compares the transaction history of the token, it cannot distinguish A ’s first transaction to B from its second. Thus B spending the token is the first occurrence that differs from the authority’s history. As described in Section IV-F, B is therefore marked as a fraudster.

There exist various solutions for preventing such an attack. One such solution is to initiate each transaction with the receiver sending a short handshake. This handshake must contain a number that is guaranteed or likely to not have been used in earlier transactions between this sender and receiver. The sender must then include this number in its transaction. If the sender attempts to replay an old transaction, the receiver can verify that the included number is different from the one specified in the handshake and refuse the transaction. To prevent the need for storing a set of previously used numbers, a pseudo-random nonce can be securely generated for each handshake.

There are, however, multiple solutions for preventing replay attacks. A suitable implementation depends on the intended use case of the digital currency and as such has been left out of scope.

H. Threat Model

In the protocol there exists one centralized authority. All other parties are clients and all clients are equal. All parties are identified by their public keys. Clients need only be online when involved in a transaction or when validating their tokens. It is assumed that clients transacting with each other can contact each other directly. All clients must contact the centralized authority to finalize their transactions. If the authority is unavailable, clients can continue transacting with each other but transactions will be unverified until they successfully contact the authority.

A malicious authority can 1) give tokens to clients at will 2) deny clients from verifying transactions and 3) conceal double spending attempts from other clients. It cannot 1) prove illegitimate accusations of double spending 2) transact in the name of its clients or 3) prevent clients from making offline transactions.

A malicious client can double spend until multiple versions of the same token reach the authority. It cannot 1) spend tokens that were never transacted to it 2) transact in the name of other clients or 3) prevent other clients from making transactions. Note that any client can see all previous recipients of its tokens until the tokens’ last verification.

I. Anonymity & Fairness

It is expected that fraudsters cannot always be penalized within the confines of the transaction system, as paying a corrective fine might not always be proportionate or possible. Penalizing fraudsters would likely require knowledge of the person or organization behind a client’s account. This is detrimental to clients’ privacy and anonymity. Finding a fair way to correct fraud and penalize fraudsters was intentionally left out of scope.

Furthermore, the described system requires aggregating a linked list of previous owners of a token, up until the last verification by an authority. Specifically, recipients of a token can see all previous recipients of that token until its last verification. This is also detrimental to privacy and anonymity. There are digital cash schemes that provide stronger notions of anonymity. Some schemes protect the identities of previous recipients and provide *unlinkability*, such that it is also impossible to relate different payments from the same client [23]. Some schemes provide an even stronger notion of anonymity where an adversary cannot recognize a token spent between other clients, even if it has already owned the token [24]. It has however been proven that an adversary can always recognize his previously-owned tokens if they are paid back to him [24].

V. IMPLEMENTATION

We prototyped the design described in Section IV. The prototype is deliberately minimalist because its function is to

provide a baseline implementation that can be evaluated. It includes only the basic facets required to transact currency per Section IV. It was implemented on the Kotlin-IPv8² protocol stack.

The IPv8 stack is typically used for connecting clients in a peer-to-peer fashion. Kotlin-IPv8’s main use case is to provide a networking overlay for clients on Android⁴ and connect them with other IPv8 clients. However, neither Kotlin-IPv8 nor our prototype rely on Android and as such it is not used during the evaluation in Section VI. Clients in IPv8 are identified by their public key and not by their IP address. Clients perform peer discovery using a gossip protocol and form *communities* for application-specific purposes. These communities are loose-knit and intended to be flexible and resistant against high churn rates. Connections are considered too fragile for TCP sessions. As such, IPv8 relies on UDP and clients connect using UDP hole punching. Kotlin-IPv8, however, is not intended exclusively for decentralized peer-to-peer communities but also provides generic communication utilities. Our prototype deploys these utilities for its purpose of transacting tokens. The specified sizes for public keys and signatures mentioned in Section IV-A were adapted from Kotlin-IPv8.

The *de facto* way of transferring binary files reliably via Kotlin-IPv8 is by using the EVA protocol [14]. Messages not sent through EVA provide no delivery guarantees. *EVA* is an acknowledgement protocol for UDP that uses acknowledgement windows to guarantee packet delivery and retransmissions. We were confronted with EVA’s limitations with regards to stability and throughput and thus opted to use our own (slightly) modified version⁵:

- Fixes a race condition that caused EVA to fail subsequent data transfers arbitrarily.
- Uses a faster and more compact encoding of lost packet numbers: $\pm 97\%$ faster serialization and $\pm 95\%$ faster deserialization.
- Allows encryption to be disabled.

Nevertheless, in Section VI-C we will demonstrate that we were unable to solve all problems with EVA that we encountered.

Furthermore, the implementation provides several scripts that were used for the performance analysis that will follow in Section VI.

VI. PERFORMANCE ANALYSIS

We present an analysis of our prototype’s performance. We performed our measurements between two processes on the same host machine, to eliminate the fluctuations of network latency. For a proper frame of reference, we place additional emphasis on low-level functionality of the underlying Kotlin-IPv8 framework. Experiments were performed on standard consumer electronics: a Lenovo Thinkpad L13 with an Intel i5 CPU operating at 2.11 GHz and 8 GB of DDR4 RAM.

⁴For Android, see <https://www.android.com>.

⁵For the modified version of EVA, refer to <https://github.com/Tribler/kotlin-ipv8/pull/71>.

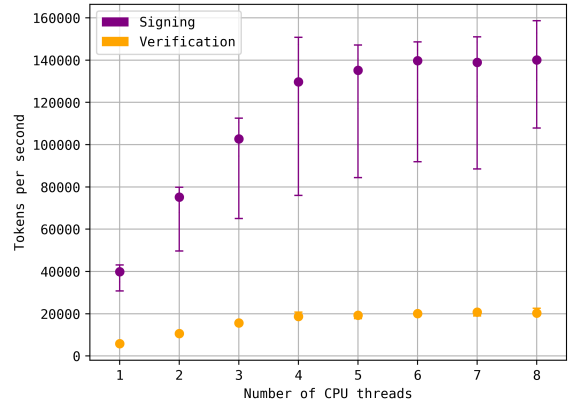


Fig. 3: The benefits of parallelism diminish quickly. Throughput of *only* cryptographic signing and verification of tokens.

A. Cryptographic Verification

We measured the throughput of cryptographic signing and verification operations to ascertain performance bounds of Kotlin-IPv8 and thus the transaction protocol. The core idea is that by stripping the implementation of all other factors, the influence of cryptographic operations on an authority’s throughput can be determined. All operations were performed with Ed25519 using a Kotlin port of Libsodium⁶, identical to Kotlin-IPv8 [25]. The chosen parameters were also identical to those used in Kotlin-IPv8.

Figure 3 shows the throughput of the cryptographic operations required from an authority to sign and verify tokens in an online setting. Per Section IV-A, signing a token requires one operation on 147 bytes. Verification requires three operations: the authority’s initial signature; the signature of the first recipient to the second; and the signature of the second back to the authority.

The figure shows that throughput increases steadily with the number of CPUs until four CPUs. Additional CPUs do not provide a significant increase in performance, which limits scalability significantly. In a practical scenario, a deployed authority would certainly require the performance benefits of parallelism beyond four CPUs. We suspect the diminishing increase to be due to resource sharing within Libsodium, although the exact reasons are unknown. Moreover, on a single CPU, the token throughput of signing and verification was only ± 38239 tokens per second (tokens/s) and ± 5840 tokens/s, respectively. Per Equation 2, this corresponds to a data throughput of only 5.62 and 2.47 *megabytes* per second (MBps).

To ascertain that the results of Figure 3 were not erroneous, we examined the throughput of cryptographic signing and verification of an authority for arbitrary binary data. Figure 4 shows the throughput of signing and verification for increasing data sizes on a single thread. Again, we used Libsodium with

⁶For Lazysodium, see <https://github.com/terl/lazysodium-java>.

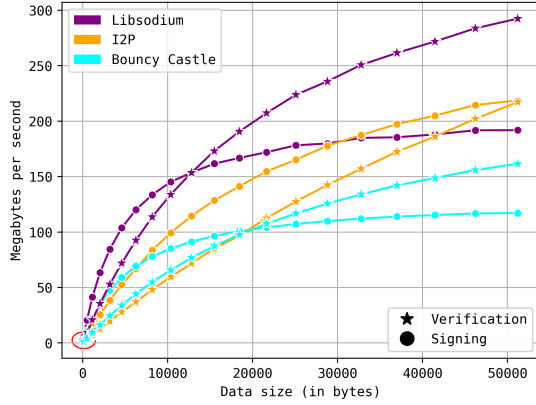


Fig. 4: Throughput of cryptographic signing and verification collapses for small data sizes. Operations on tokens are at most 147 bytes, marked by the red circle.

Ed25519 to mimic Kotlin-IPv8. We also measured the performance of Ed25519 on two other implementations, Bouncy Castle⁷ and I2P⁸. The figure shows that larger file sizes are tremendously faster to sign and verify than smaller, for all implementations. It also shows that, by comparison, Libsodium has a relatively fast implementation of Ed25519.

Per Figure 4, signing has a throughput of ± 6.05 MBps for files of 147 bytes. Signing 6.05 MBps is equivalent to signing 41156 tokens/s. According to Figure 3, signing tokens on 1 thread has a throughput of ± 38239 tokens/s. The throughput of a verification operation on 147 bytes is 2.60 MBps, per Figure 4. Verification of a token requires three operations of 147, 138, and 138 bytes. For simplicity, we assume an upper bound of 147 bytes per operation. Thus, per Figure 4, the throughput of verifying tokens would be $\pm \frac{2600000}{3 \cdot 147} = 5896$ tokens/s. In Figure 3, verifying tokens on 1 thread has a throughput of ± 5840 tokens/s. We thereby conclude that the measurements obtained in Figure 3 were not erroneous.

Figure 5 shows the throughput of cryptographic verification of an authority for an increasing number of offline recipients. The case where the number of recipients equals 2 corresponds to the online setting.

B. Data Transfer

Figure 6 shows the throughput of data transfers on a range of different software layers. The majority of these layers have no practical usage for our prototype. They do, however, provide valuable insights in its throughput and that of Kotlin-IPv8. We measured a native pipeline as the upper bound of throughput on our host machine (± 1652.28 MBps). Then, we measured local UDP traffic (± 801.16 MBps). All subsequent measurements use UDP and were performed on Kotlin, which executes on the Java Virtual Machine. We first

⁷For Bouncy Castle, see <https://github.com/bcgit/bc-java>.

⁸For I2P, see <https://github.com/i2p/i2p.i2p>.

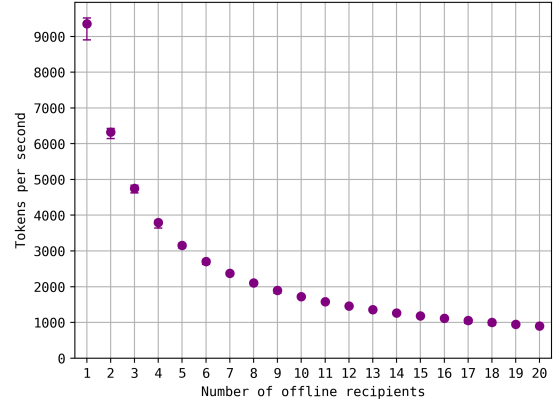


Fig. 5: Throughput of token verification declines for additional recipients.

performed a measurement where we reused one UDP datagram in memory (± 652.16 MBps). Then, we performed the same measurement but recreated datagrams for each transmission (± 560.29 MBps). Subsequent measurements were performed on Kotlin-IPv8 (± 30.71 MBps). Figure 6 shows that switching to Kotlin-IPv8 incurs a significant drop in throughput. For our next measurements, we enabled Kotlin-IPv8’s encryption feature, which performs per-packet encryption (± 14.99 MBps). Finally, we measured the throughput of EVA for its most performant configuration with encryption enabled (± 8.25 MBps).

Figure 6 shows that additional software layers decrease throughput. Most of these layers are necessary and outside the scope of this thesis, and the respective drops unpreventable. Within the scope of this thesis are the measurements regarding Kotlin-IPv8 and EVA. Kotlin-IPv8 incurs the largest percentual drop of throughput in the figure, $\pm 95\%$. Applications built on Kotlin-IPv8 will invariably be limited by this.

Due to further unexpected results, we have detailed EVA performance in Section VI-C.

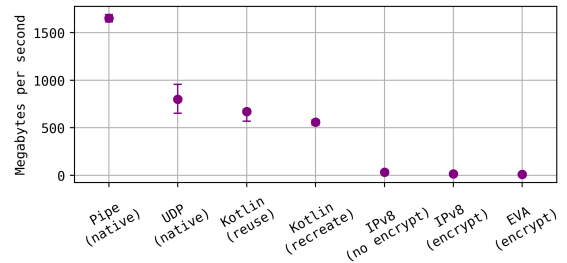
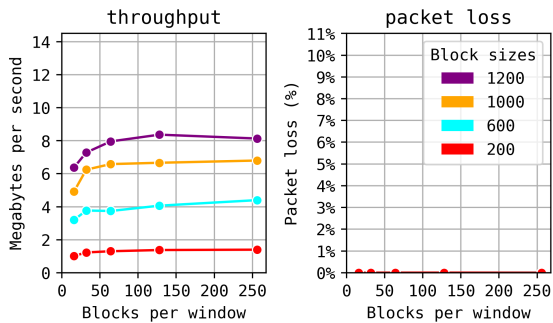


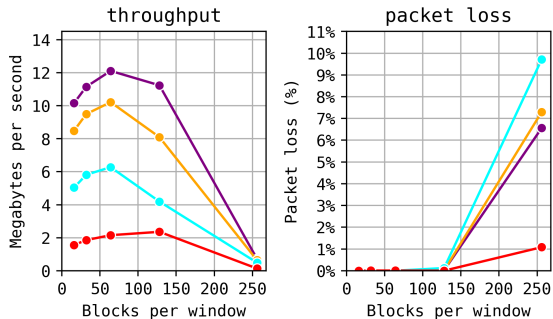
Fig. 6: Throughput of various data transfer methods.

C. EVA Acknowledgement Protocol

Figure 7 shows EVA’s throughput and packet loss for various configurations. EVA’s main configurable parameters are the number of payload bytes per UDP packet (*block*



(a) Throughput and packet loss with encryption enabled.



(b) Throughput and packet loss with encryption disabled.

Fig. 7: Throughput and corresponding packet loss of EVA with encryption enabled (Figure 7a) and disabled (Figure 7b). Packet loss soars for increasing *blocks per window* when encryption is disabled. UDP buffers were fixed at 106496 bytes.

size) and the number of blocks per acknowledgement window (*blocks per window*). There is a positive correlation between throughput and both *block size* and *blocks per window*, save for the last two measurements of Figure 7b. From Figure 7b it can also be observed that all measurements of 256 *blocks per window* show subpar throughput when encryption is disabled.

To explain the sudden drop in throughput, Figure 7 also shows the measurements’ corresponding packet loss. From the figures it can be observed that packet loss increases as throughput decreases. Our measurements were performed on a single host, so external routing devices cannot account for the anomaly. The fact that none of the measurements in Figure 7a (where encryption is enabled) suffer from packet loss indicates that processing speed is not the bottleneck either; enabling encryption is strictly more intensive than not. We therefore hypothesized the drop to be caused by UDP buffers overflowing.

Figure 8 shows that this is indeed the case. We fixed *blocks per window* at 256 (corresponding to the slowest measurements in Figure 7) and repeated our measurements for varying UDP buffer sizes. The leftmost measurement shows results for a buffer size of 106k bytes, our initial configuration. The figure shows that packet loss decreases as *buffer size* increases. In turn, there is a positive correlation between *buffer*

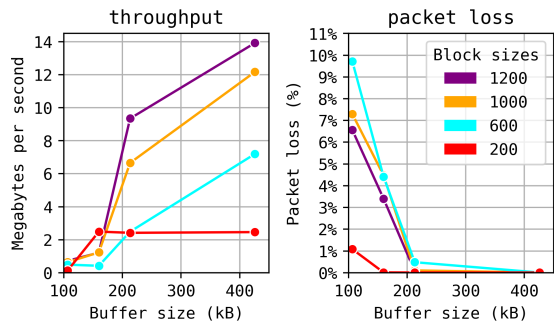


Fig. 8: EVA’s throughput increases and its corresponding packet loss decreases when UDP buffers are expanded. The number of *blocks per window* was fixed at 256, corresponding to the slowest measurement in Figure 7. Encryption is disabled.

size and throughput. It can also be observed that as packet loss nears 0%, increasing *buffer size* has no effect.

From inspection of the source code, we found that Kotlin-IPv8’s EVA implementation attempts to transmit an entire window’s worth of data at once. For example, at 256 blocks per window and blocks of 1200 bytes, this results in a series of consecutive transmissions amounting to $256 \cdot 1200 = 307.2\text{kB}$ excluding overhead. This transmission causes buffer overflows and thereby packet loss. We found that enabling encryption acts as an unintended form of *flow control*. Due to reduced throughput, the buffer does not fill up. We have left improving EVA with proper flow control out of scope.

D. End-to-end Token Throughput

Table I shows the end-to-end throughput of the prototype for one thread in an online setting, in tokens per second. The measurements show a full round trip of a token, from its signing to its validation. This process is displayed in sequential

TABLE I: End-to-end token throughput

	Tokens/s	MBps	Signatures/s
Authority signs tokens for recipient #1	37663	-	-
Recipient #1 receives tokens from authority	43523	12.49	-
Recipient #1 validates tokens from authority	17477	-	17477
Recipient #2 receives tokens from recipient #1	34223	14.54	-
Recipient #2 validates tokens from recipient #1	8797	-	17594
Authority receives tokens from recipient #2	23452	13.20	-
Authority validates tokens from recipient #2	5143	-	15429

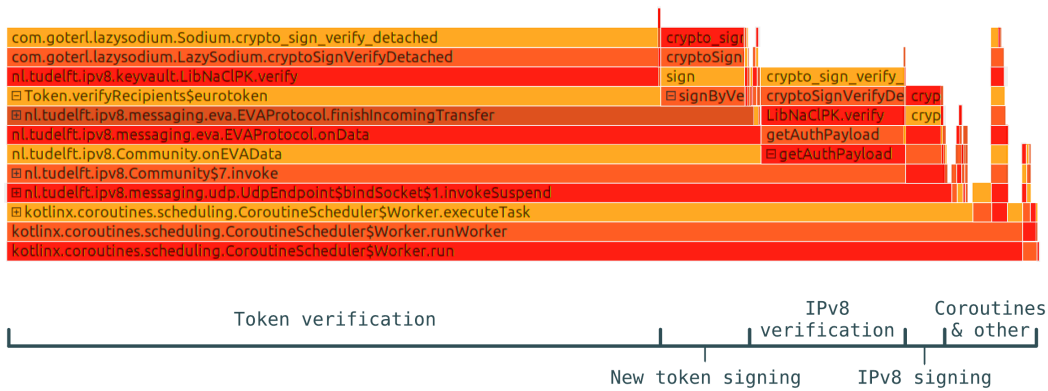


Fig. 9: Flame graph of an authority verifying tokens.

order from top to bottom. From the table, it can be observed that the throughput of transfer and verification declines as the tokens grow. As defined by Equation 1, each transaction adds 138 bytes to a token. However, the number of signatures verified per second (*signatures/s*) is relatively constant. The majority of the execution time in the verification process goes to verifying signatures. Thus, if a token grows with a factor N , it will be $\pm N$ times slower to verify. Therefore, we can establish that with our setup, the authority’s verification throughput is at most roughly ± 17000 signatures per second per thread, regardless of the size of the tokens it verifies.

E. Verification Flame Graph

Figure 9 shows a flame graph of an authority verifying tokens and returning them to their owner [26]. From the figure it can be observed that token verification was accountable for $\pm 63\%$ of the workload on the CPU. Verifying the cryptographic signatures of incoming IPv8 packets accounted for $\pm 14\%$ of the workload. Per Section IV-E, the authority returns verified tokens to their owners. From Figure 9 it can be observed that signing these tokens accounted for $\pm 8\%$, and signing the resulting IPv8 packets accounted for $\pm 4\%$ of the workload. The remaining $\pm 11\%$ was accounted for by coroutine scheduling, logging, and miscellaneous tasks.

From the results it is apparent that cryptographic signing and verification dominate the computational load of an authority. In total, cryptographic operations accounted for $\pm 89\%$ of the workload. To increase an authority’s throughput, it is therefore crucial to accelerate and parallelize cryptography first and foremost. Currently, parallelization of cryptography is limited, as shown in Figure 3.

VII. CONCLUSION

This thesis describes a token-based transaction protocol and its implementation on the Kotlin-IPv8 protocol stack. The protocol allows funds to be spent in an offline setting and provides retroactive fraud detection. Fraud can be detected because each token maintains a list of its previous recipients. Clients can submit their tokens to a monetary authority for verification. Upon verification, the authority stores the list of previous

recipients. Each time a token is sent in, its list is compared to the list stored by the authority. Discrepancies between the lists prove that double spending was done and reveal by whom. Many facets of the protocol require improvement before it can be used as a practical digital currency. The protocol is not intended for deployment but instead serves as a trial for building digital currencies on Kotlin-IPv8.

The performance analysis of our prototype emphasizes various areas where Kotlin-IPv8 requires improvements before it can support a highly scalable digital currency. The analysis shows that:

- 1) The throughput of cryptographic operations does not scale with the number of available CPUs.
- 2) Adding Kotlin-IPv8 as a semantic layer on top of Kotlin’s own UDP stack reduces throughput by $\pm 95\%$.
- 3) Kotlin-IPv8’s implementation of the EVA protocol overlooks the system’s available UDP buffers and experiences massive packet loss for certain configurations.

Nevertheless, the performance analysis also shows that cryptographic signing and verification account for the majority of an authority’s computational load. Thus, for improving an authority’s verification throughput in the current prototype, we recommend accelerating its cryptographic operations.

REFERENCES

- [1] ECB, “Report on a digital euro,” ECB, Tech. Rep., 2020. [Online]. Available: https://www.ecb.europa.eu/pub/pdf/other/Report_on_a_digital_euro~4d7268b458.en.pdf
- [2] F. Panetta, “Public money for the digital era: towards a digital euro,” 2022, keynote speech by Fabio Panetta, Member of the Executive Board of the ECB. [Online]. Available: <https://www.ecb.europa.eu/press/key/date/2022/html/ecb.sp220516~454821f0e3.en.html>
- [3] M. McGuinness, “Speech by commissioner mcguinness at the european parliamentary financial services forum on a digital euro,” 2023, speech by Mairead McGuinness, European Commissioner for Financial Stability, Financial Services, and the Capital Markets Union. [Online]. Available: https://ec.europa.eu/commission/presscorner/detail/en/SPEECH_23_1547
- [4] ECB, “Call for expression of interest for digital euro front-end prototyping,” 2022, a call for expression of interest by ECB regarding a front-end prototype for a digital Euro. [Online]. Available: https://www.ecb.europa.eu/paym/digital_euro/investigation/profuse/shared/files/dedocs/ecb.dedocs220428.en.pdf

- [5] —, “Faq’s on the digital euro,” 2022, an FAQ by ECB for the digital Euro project. [Online]. Available: https://www.ecb.europa.eu/paym/digital_euro/faqs/html/ecb.faq_digital_euro.en.html
- [6] N. Jonker, L. Hernandez, R. de Vree, and P. Zwaan, “From cash to cards: how debit card payments overtook cash in the netherlands,” DNB, Tech. Rep., 2018. [Online]. Available: https://www.dnb.nl/media/kx1akmnb/201802_nr_1_-2018-_from_cash_to_cards_how_debit_card_payments_overtook_cash_in_the_netherlands.pdf
- [7] N. Jonker and P. Zwaan, “Betalen aan de kassa 2020,” DNB, Tech. Rep., 2020. [Online]. Available: https://www.dnb.nl/media/e34bo5zu/betalen_kassa_2020.pdf
- [8] S. Riksbank, “Pandemic hastening development towards digital payments,” 2021, an article published by Sveriges Riksbank about the declining usage of cash in Sweden and other countries. [Online]. Available: <https://www.riksbank.se/en-gb/payments--cash/payments-in-sweden/payments-report-2021/1.-trends-on-the-payment-market/pandemic-hastening-development-towards-digital-payments/cash-used-less-and-less-frequently-in-sweden-and-abroad/>
- [9] ECB, “The paradox of banknotes: understanding the demand for cash beyond transactional use,” *ECB Economic Bulletin, Issue 2/2021*, 2021. [Online]. Available: https://www.ecb.europa.eu/pub/economic-bulletin/articles/2021/html/ecb.ebart202102_03~58cc4e1b97.en.html
- [10] Office of Science and Technology Policy, “Technical evaluation for a u.s. central bank digital currency system,” Office of Science and Technology Policy, Tech. Rep., 2022. [Online]. Available: <https://www.whitehouse.gov/wp-content/uploads/2022/09/09-2022-Technical-Evaluation-US-CBDC-System.pdf>
- [11] Working Group on E-CNY Research and Development of the People’s Bank of China, “Progress of research & development of e-cny in china,” Working Group on E-CNY Research and Development of the People’s Bank of China, Tech. Rep., 2021. [Online]. Available: <http://www.pbc.gov.cn/en/3688110/3688172/4157443/4293696/2021071614584691871.pdf>
- [12] House of Lords, “Central bank digital currencies: a solution in search of a problem?” 2022. [Online]. Available: <https://publications.parliament.uk/pa/ld5802/ldselect/ldconaf/131/131.pdf>
- [13] National Commission on Informatics and Liberty, “Digital euro: what is at stake for privacy and personal data protection?” 2022. [Online]. Available: <https://www.cnil.fr/en/digital-euro-what-stake-privacy-and-personal-data-protection>
- [14] J. Bambacht and J. Pouwelse, “Web3: A decentralized societal infrastructure for identity, trust, money, and data,” Master’s thesis, Delft University of Technology, 2022.
- [15] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” *Decentralized Business Review*, p. 21260, 2008.
- [16] S. Canard, D. Pointcheval, O. Sanders, and J. Traoré, “Divisible e-cash made practical,” in *IACR International Workshop on Public Key Cryptography*. Springer, 2015, pp. 77–100.
- [17] D. Chaum, “Blind signatures for untraceable payments,” in *Advances in cryptology*. Springer, 1983, pp. 199–203.
- [18] T. Okamoto and K. Ohta, “Disposable zero-knowledge authentications and their applications to untraceable electronic cash,” in *Conference on the Theory and Application of Cryptology*. Springer, 1989, pp. 481–496.
- [19] M. Stadler, J. Piveteau, and J. Camenisch, “Fair blind signatures,” in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1995, pp. 209–219.
- [20] R. Blokzijl, “Eurotoken,” Master’s thesis, Delft University of Technology, 2021. [Online]. Available: <http://resolver.tudelft.nl/uuid:132faae8-6883-454f-a8ce-94735340dce9>
- [21] V. Hajric and K. Greifeld, “Bitcoin went mainstream in 2021. it’s just as volatile as ever,” Bloomberg, 2021. [Online]. Available: <https://www.bloomberg.com/graphics/2021-bitcoin-volatility/>
- [22] D. Bullmann, J. Klemm, and A. Pinna, “In search for stability in crypto-assets: are stablecoins the solution?” ECB, Tech. Rep., 2019. [Online]. Available: <https://www.ecb.europa.eu/pub/pdf/scpops/ecb.op230%7Ed57946be3b.en.pdf>
- [23] S. Canard, A. Gouget, and J. Traoré, “Improvement of efficiency in (unconditional) anonymous transferable e-cash,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2008, pp. 202–214.
- [24] S. Canard and A. Gouget, “Anonymity in transferable e-cash,” in *International Conference on Applied Cryptography and Network Security*. Springer, 2008, pp. 207–223.
- [25] D. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Yang, “High-speed high-security signatures,” *Journal of cryptographic engineering*, vol. 2, no. 2, pp. 77–89, 2012.
- [26] B. Gregg, “The flame graph,” *Communications of the ACM*, vol. 59, no. 6, pp. 48–57, 2016.