Digital Offline Euro - First Realisation

Robbert Koning
dept. name of organization (of Aff.)
Delft University of Technology
Delft, The Netherlands
R.M.Koning@student.tudelft.nl

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

In recent years, the European Central Bank (ECB) has increased its efforts in exploring the possibility of realising its own Central Bank Digital Currency (CDBC), the 'digital Euro'. The ECB has published various reports and resources that outline the desirability, necessity even, of such a project (i.e. [1], [2]). Calls for expression of interest are being published and the ECB is in its investigation phase until October 2023 [3] [4].

Most resources mention the decline of cash usage and corresponding rise of digital payments as a prominent reason for a digital Euro. 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 [5] [6]. Cash is currently the only publicly accessible form of sovereign money [2]. Digital payments are made using services provided by private and/or foreign (non-European) actors. The money involved in these transactions is a liability of the respective actor and not a claim on a European central bank.

Perhaps more importantly, a report published by ECB discusses a potential 'currency substitution'; a scenario where a new form of money that is entirely unregulated by ECB becomes a viable medium of exchange and store of value. Currency substitution could reduce the effectiveness of ECB's monetary policy, harm market competition, and finally even threaten the European Union's strategic independence [1]. The private and/or foreign actors that are largely responsible for the fear of currency substitution are large corporations, big tech, and foreign central banks [2] [7]. In order to compete with these parties, the ECB has enumerated many requirements and wishes for its CBDC. First and foremost it is necessary for the value of digital Euros to be anchored to physical Euros. Moreover, the ECB wishes for its CBDC to enjoy beneficial cash-like features, such as being usable in an offline setting. For a full specification of the ECB's requirements and wishes for its CBDC, we refer the reader to the report [1].

Some of the demands and wishes mentioned by the ECB are difficult to realise individually and perhaps not even unifiable. The aforementioned report outlines multiple scenarios and analyses [1]. This research focuses on a scenario that attempts to somewhat resemble cash usage; physical Euros are mimicked by digital units of fixed, indivisible value ('tokens') and emphasis is placed on researching their functioning in an offline setting. Due to technical limitations however, some design choices were made that do not fall in line with the anonymity and decentralisation of cash usage. Please refer to Section VI for further elaboration. This research contributes 1) a token-based transaction system 2) a performance analysis of various bottlenecks in the system to highlight its weaknesses and empirical upper performance bounds and 3) a software-tested reference implementation.

II. Problem Description

This research focuses on exploring offline spending in CBDCs. Various other problems that trouble common CBDC design are discussed in Section VI and/or are left out of scope. 'Offline spending' in this context refers to the action of spending a CBDC without having a connection to its network. This is crucial in areas without a reliable internet connection or in case of network/system failure.

A prominent example of currency that can be spent offline is cash. Cash, however, has various other drawbacks, which is reflected in its declining usage over the last years [6]. Cash is passed around physically, which is crucial to its offline functioning. In a cash transaction, the sender passes their currency to the receiver, thereby ensuring that they are unable to spend their currency again because they do not possess it anymore. Trivial as it may seem, for digital systems this is difficult to realize. In both decentralized and offline transaction systems it is non-trivial to verify whether parties still own the funds they want to spend and have not spent them before. Unlike cash, which is designed to be difficult to recreate, digital data can be copied easily. Thus verifying transactions requires some information about the sender - for instance whether they have spent the involved funds or not - though details differ per protocol. In a decentralized environment it is difficult to obtain this information reliably from peers due to potential malicious behavior. In an offline environment, connectivity problems further increase this difficulty.

With a toy example in an offline setting, we envision a malicious party A who transacts their digital currency to honest party B. After this transaction, A can simply 'undo' their last transaction, for instance by keeping copies of the funds they

transacted to *B* and restoring them. Now *A* can transact the same funds they already spent to honest party *C*. Without an additional direct or shared connection from *B* to *C*, the honest parties cannot determine whether *A* committed fraud. This is called the 'double spending problem' and we assume the reader to be familiar with it [8].

There have been numerous attempts at solving or mitigating the double spending problem. A large fraction of the proposed solutions utilize a form of 'global consensus', which requires connectivity with the rest of the network. Global consensus disallows offline transfers and is therefore not robust enough to substitute cash.

III. Related Work

A. Eurotoken

We consider the main prior work for this thesis to be Eurotoken [9]. Most of our design decisions are based upon the strengths and weaknesses of this work.

One such decision is to have a limited number of trusted authorities to verify transactions. This makes the system distributed but not decentralized. The advantage of this approach is that it enables the respective central bank to enforce potential future policies. Moreover, it provides a non-deterministic near-immediate transaction finality.

A crucial lesson from this work is that balance-based systems appear to greatly complicate robustness measures. We believe that a token-based architecture is more resilient and easier to realise. A token-based system requires the generation of tokens - analogous to minting coins - and a different transaction protocol. The token minting process and transaction protocol are described in Section IV. A major implication of a token-based system is that multiple tokens need to be sent per payment, comparable to how cash payments often require multiple notes and coins.

From measurements it became apparent that Eurotoken's transaction throughput was not high enough to facilitate the needs of the Eurozone. Transactions were measured to be around ?. VISA is capable of processing 24000 transactions per second (self-proclaimed, [10]) and Alipay 544000 [11]. It is worth noting that the scale of these systems is massively larger than measured in the Eurotoken paper, which results in skewed measurements [9].

IV. Design and Architecture

This research proposes a centralized CBDC that allows offline transactions with fixed-value tokens and guarantees retroactive fraud detection.

The proposed system requires a number of trusted parties that are in charge of token exchange and transaction verification. We refer to these parties as 'authorities' and identify them by their public key. The limited number of trusted authorities makes verification a distributed yet centralized operation. The motivation for this design choice is elaborated upon in Section III-A. The process of exchanging currency for tokens is beyond the scope of this paper and is briefly discussed in Section VI.

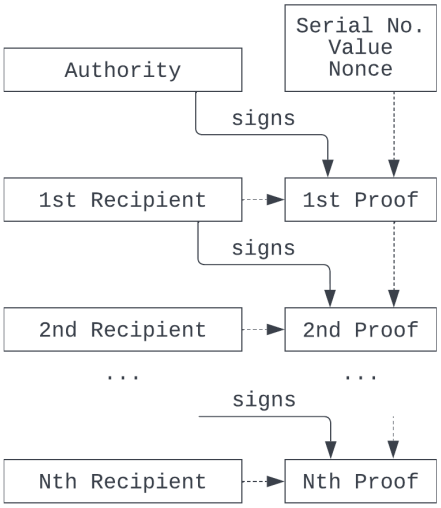

Fig. 1. Graphical representation of a token. Tokens represent monetary units of fixed value that store all their previous recipients until they are verified by an authority.

Clients are all system participants that are not authorities. They, too, are identified by their public key. It is assumed that clients know the public keys of the authorities in the network. It is also assumed that authorities know the real identities of clients. While this is not necessary for the proposed system to function, implicating a public key with fraud loses its severity if the instigator can remain anonymous. This is discussed further in Section VI-A.

Clients can transact tokens to each other and consult authorities to verify the validity of their tokens. If clients cannot connect to an authority, for instance during a power outage, they can continue transacting but defer verification until they can connect.

To realize retroactive fraud detection, the proposed 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]:

1) *Serial number*. An 8-byte unique token identifier.
2) *Value*. A 1-byte representation of the token's worth in Euros. Like cash, tokens have a limited number of fixed denominations and the byte values are mapped to those. For example, a token worth 1 Euro has a byte value of 7, though this mapping is arbitrary.
3) *Authority public key*. A 74-byte public key of the authority that is in charge of the token (the 'authority').

[1]The bit-lengths of the signatures and public keys were adapted from those used in Kotlin-IPv8 (https://github.com/Tribler/kotlin-ipv8), upon which the implementation was built, and are not integral to the protocol's functioning.

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 key 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*. Likewise, all subsequent pairs follow the same pattern; they contain a signature by 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.

B. Token Minting

When a token is created, its *Serial number*, *Value*, *Nonce*, *Authority public key*, and *Recipients* list 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 the *Recipients* list) 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 the *Recipients* list. 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 proof 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 proof 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* list that contains the desired recipient's public key and a signature of the token's last proof together with the desired recipient's public key. This is depicted in Figure 1.

E. Authority Verification

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

1) The authority ensures that the received token has more than 1 recipient in its *Recipients* list. If not, the token is either invalid or ineligible for verification.

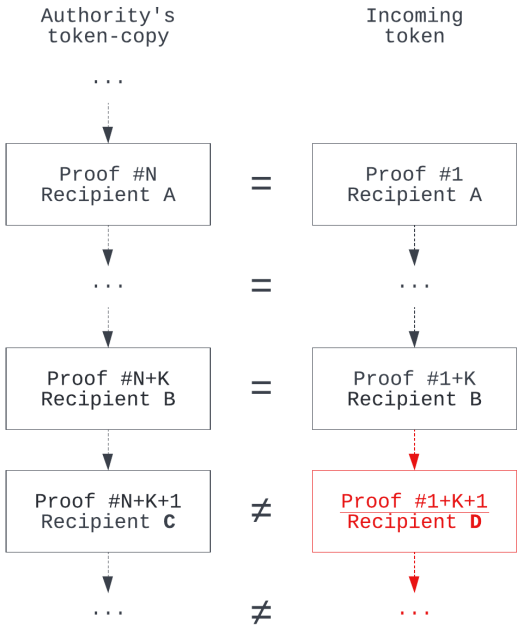

Fig. 2. The authority's double spending detection mechanism. In the figure, recipient B doubly spent a token, which was detected because proof $N+K+1$ of the authority was not equal to proof $1+K+1$ of the incoming token. This mechanism can only be applied if the authority's token-copy's last proof is not equal to the incoming token's *First proof*.

2) The authority ensures that the token's last 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 itself is the authority that manages the token. Then it verifies that the token is still in circulation.
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 by comparing the proof of the last pair ('last proof') of its token-copy to *First proof* in the received token. If these are identical, double spending cannot be proven (see Section IV-F) and the authority will finalize verification. Finalizing verification requires the authority to update its copy of the token by appending all new recipient-proof pairs of the received token to its *Recipients* list. It will also append a new pair containing the desired recipient - the one who sent the token for verification - and a corresponding proof.
6) The authority sends the verified token to the desired recipient.

F. Double Spending Detection

In Section IV-E it is mentioned that the authority updates its token-copy's *Recipients* list upon a valid verification. This

means that its last proof is updated as well. To detect double spending, an authority compares the last proof of its token-copy to *First proof* in the received token. A diagram of this scenario is depicted in Figure 2.

If a token is doubly spent, then multiple versions of the token will eventually reach their authority. The first time, 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 does not correspond to the doubly spent token's *First proof* anymore. Thus, if the proofs differ double spending has occurred. If the proofs are equal, double spending might have occurred.

When double spending is detected, the authority will search for the instigator. It will find the received token's *First proof* in the *Recipients* list of its token-copy. It will then compare the recipient-proof pairs of the token-copy with those of the received token starting from the pairs that contain *First proof* in both lists, respectively. 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 proof that double spending was performed by the client that signed them.

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 a transaction with the receiver sending a short handshake that includes a pseudo-random nonce. The sender must include this nonce in its transaction to proof with overwhelming probability that they did not replay the transaction. Another solution is to have receivers maintain a list of the last proofs of all tokens they have ever received.

H. Tokens vs. Balances

The choice to use tokens instead of balances was motivated by the protocol's emphasis on supporting offline transactions. Integral to the effectiveness of this protocol is another that, given a proof of fraud, provides conflict resolution and damage mitigation. It thereby deters fraud without the need to 'solve' the double spending problem in an offline setting. The protocol described in Section IV provides a way for authorities to eventually proof fraud has occurred, even in offline settings, on a per-token basis. This proof only requires knowledge that pertains to a single token. Generalizing such a proof to a balance, which is an aggregation of multiple transactions, requires a more complex proof. Although we cannot give conclusive evidence regarding the difficulty of such a proof, we consider it to be beyond the scope of this paper.

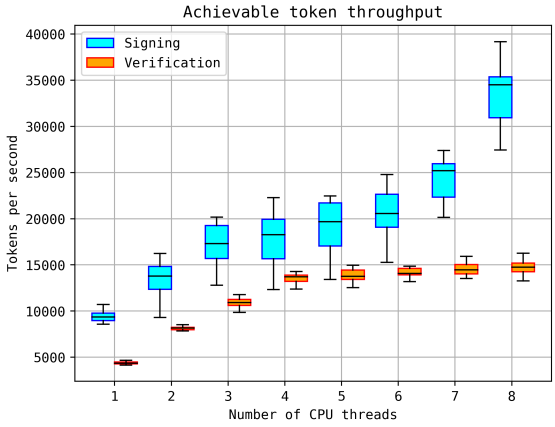

Fig. 3. Throughput of *only* cryptographic verification and signing of tokens.

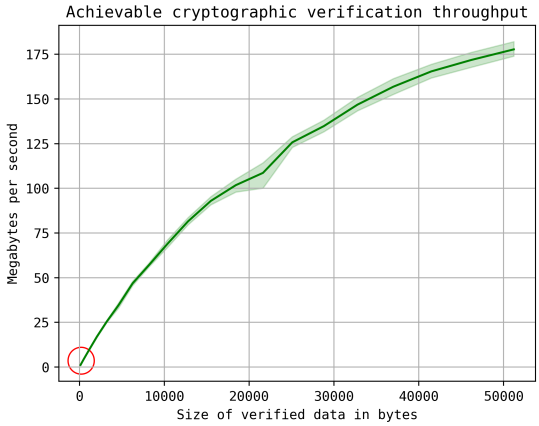

Fig. 4. Throughput of cryptographic verification for varying data sizes *on a single CPU thread*. The red circle marks 201 bytes, the size of a token.

V. Performance Analysis

We analyzed the system's performance to expose its shortcomings. For a proper frame of reference, we also performed a brief performance analysis of low-level functionality such as data transfer throughput and cryptographic operations.

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. All experiments were performed 5 times.

A. Cryptographic Verification

We measured the throughput of various cryptographic operations to ascertain the upper performance bounds of our protocol. The core idea is that by stripping the implementation of all other factors, we can determine the influence of cryptographic operations on the authority's overall throughput. All

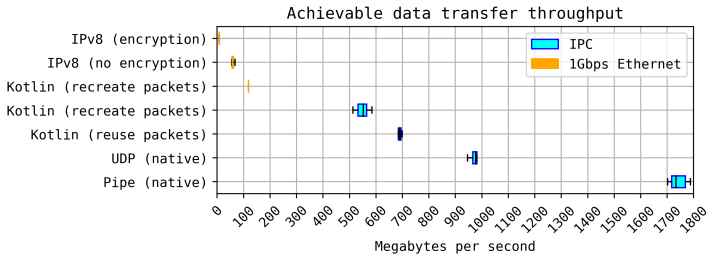

Fig. 5. Throughput of various data transfer methods.

operations were performed with Ed25519 using a Kotlin port of Libsodium[2] [12]. The chosen parameters were identical to those used in IPv8[3].

Figure 3 shows the throughput of the cryptographic operations required to verify tokens in an online scenario. As described in Section IV, the authority's signature needs to be verified as well as the first recipient's. Figure 3 shows that throughput increases monotonically although not linearly with the number of CPUs, even though verification and signing processes can be executed independently from each other. We suspect the diminishing increase to be due to resource sharing, although the exact reasons are unknown. Interestingly, the highest verification measurement of 17483 tokens per second, at 201 bytes to verify per token, corresponds to a signature verification throughput of only 3.51 megabytes per second. To verify this was not an erroneous result, we measured signature verification for different data sizes.

Figure 4 shows the throughput of cryptographic verification for varying data sizes on a single thread. It is apparent that larger file sizes are tremendously faster to verify than smaller. We expect this to hold true for signing operations well. A potential improvement to the protocol that can utilize this fact is described in Section VI.

B. Data Transfer

EVA's implementer could not give the exact reason for EVA's observed low throughput but speculated it to be a limitation of the underlying IPv8 protocol [BAMBACHT]. To verify this assumption, we performed additional measurements that we show in Figure 5. Figure 5 shows that the overhead of using Kotlin as opposed to a natively compiled UDP sender is significant but not problematic. Kotlin maximally utilizes the available bandwidth when constrained by a 1Gbps connection, measuring a throughput of almost 125 megabytes per second. The overhead of Kotlin-IPv8 is indeed significant, dropping to an average of 60.2 megabytes per second without encryption and 8.3 with. When encryption is enabled, each individual IPv8 packet is encrypted. Based upon the results of Figure 4, we expect encryption to also be a bottleneck for packet throughput. Nevertheless, even encrypted IPv8 traffic was massively faster than EVA's throughput for all measured configurations. This eliminates IPv8 as a potential reason for EVA's low throughput.

[2]For Lazysodium, see https://github.com/terl/lazysodium-java.

[3]For Kotlin-IPv8, see https://github.com/Tribler/kotlin-ipv8.

VI. Discussion & Future Work

In the vast design space of CBDCs, this research focuses specifically on offline usage. Realizing even limited offline functionality came with a cost, meaning we've had to compromise on other desirable features.

A. Anonymity

For offline usage, the proposed system requires aggregating a linked list of previous owners of a token, up until the last verification by an authority. Unless the means of identifying past owners - currently by means of a public key - can be decoupled from the owners' real identities, this has enormous implications on users' privacy. Specifically, recipients of a token can see all previous recipients of that token until its last verification. It is expected that exposing this list of owners is unnecessary, although a solution is as of yet unknown.

Moreover, it is assumed that authorities know the identities of their clients. It is expected that fraudsters cannot always be penalized within the confines of the transaction system. For example, dealing a corrective fine requires a convict to own enough tokens to pay. If a fine cannot be paid, corrective actions need to be taken in another way that does not involve tokens. Finding a fair way to correct fraud and penalize fraudsters was deliberately left out of scope.

B. Decentralization

The system depends on a number of trusted authorities to verify transactions. If the system is deployed as a substitute for cash, then decentralization is desirable. Decentralizing the system would likely have disadvantages that might be unacceptable, such as probabilistic transaction finality, limited scalability, and less effective monetary policy. For this work, we opted to choose the same approach as our main prior work [9].

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 [13]. There are various ways in which the value of an asset can be kept stable and this topic has gained renewed interest with the rise of 'stablecoins' - cryptocurrencies that aim to be non-volatile with regards to a major non-cryptocurrency. There is an inverse relationship between the potential stability of stablecoins and how much they are decentralized. The strongest stabilization mechanism is collateralisation 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 decentralisation and stability [14]. To the best of our knowledge, no decentralised and highly stable stablecoins exist.

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] 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

[4] ——, "Faqs 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

[5] 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

[6] 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

[7] BIS, "G7 working group on stablecoins," BIS, Tech. Rep., 2019. [Online]. Available: https://www.bis.org/cpmi/publ/d187.pdf

[8] U. Chohan, "The double spending problem and cryptocurrencies," *SSRN*, 2021.

[9] R. Blokzijl, "Eurotoken," Master's thesis, Delft University of Technology, 2021. [Online]. Available: http://resolver.tudelft.nl/uuid:132faae8-6883-454f-a8ce-94735340dce9

[10] VISA, "Visa acceptance for retailers." [Online]. Available: https://usa.visa.com/run-your-business/small-business-tools/retail.html

[11] J. Zhang, "How alibaba powered billions of transactions on singles' day with 'zero downtime'," South China Morning Post, 2019. [Online]. Available: https://www.scmp.com/tech/e-commerce/article/3038539/how-alibaba-powered-billions-transactions-singles-day-zero-downtime?module=perpetual_scroll_0&pgtype=article&campaign=3038539

[12] 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.

[13] 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-volitility/

[14] 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

Acknowledgment

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.