Stronger Promises
A Privacy-Preserving, Post-Quantum Credit and Repayment System

Alan Szepieniec[1], Taariq Lewis[2], and Giuseppe Ateniese[3]

[1] imec-COSIC, KU Leuven, Belgium
[2] Promise `https://promiseprotocols.com/`
[3] Stevens Institute of Technology, Hoboken, New Jersey

Abstract. Promise is a digital currency protocol introduced by Yoshiro in 2018 [15] to disintermediate third-party payment processors specifically in the context of payment processing and credit reputation. The present paper introduces Stronger Promises, an upgrade to the aforementioned Promise protocol. With this upgrade, Promise provides private transactions with post-quantum security that interface with The Pledge—Promise's native payment contract script—in a way that marries access control with privacy to allow any authorized or independent proxy to complete contract repayments under any payer's public key. Additional features of the upgraded Promise network include a virtual machine for smart contracts, and a reputation system to incentivize node participation.

Keywords: cryptocurrency · privacy · post-quantum · payment processing

1 Introduction

In 2018, Yoshiro Shinji published a paper on a new protocol called Promise [15], which aimed to be a "Decentralized, Peer-to-Peer Proxy Repayment Protocol." The objective of that paper was to explore the creation of a new transaction type and payment system to replace the servicing activity of third-party payment processing companies. Third-party payment processors act as service intermediary between parties in a financial contract. By acting as a central routing and collection system, these intermediaries power most of the everyday payment interactions in digital payment systems. In Bitcoin and other peer-to-peer digital cash systems, individuals are expected to create individual transactions with each other, for each payment. This type of individual payment processing is expensive in both time and effort as they require each participant to enter into and manage each payment agreement with each other.

Although the original Promise protocol introduced innovative cryptographic primitives to enable the new payment processing features, it did leave unanswered a number of important challenges that arise from a view to a long-term and globally scalable payments system.

- Promise's digital signatures were based on elliptic curve cryptography, which is not designed to withstand attacks performed on quantum computers. Given the progress in the construction of quantum computers, a digital currency aiming to protect its value in a time span of ten to twenty years must be designed with **quantum-resistant cryptography** at its base.
- Promise's proxy re-signing features did not allow for **private transactions** separate from the protocol's repayment contracts.
- Promise's contract data type did not allow for **authenticated payments** into its native contract, thus exposing it to unauthorized payments which could make credit reputations harder to build in an untrusted network.

1.1 Introducing Stronger Promises

We propose Stronger Promises, an improved Promise protocol featuring long-term secure privacy-preserving transactions integrated with Promise's dedicated active data structure, called the *Pledge*, which doubles as a pre-set template for repayment contracts on the one hand, and an immutable record of users' repayment history on the other. Users of the Pledge contract to receive and pay aggregate payment flows from a number of participants. The Pledge allows users to cultivate and determine reputation, participate on either side of the Pledge, trade Pledge obligations or rights, or link Pledge payments into a web of automatic cashflow chains.

Promise relies exclusively on post-quantum cryptographic primitives. The anticipated arrival of large-scale quantum computers will therefore barely affect the security of the Promise network. In contrast, the same quantum breakthrough will break the ECDSA signature scheme and kill all cryptocurrencies that rely on it, including Bitcoin, Ethereum, *etc.*

The privacy of standard transactions is guaranteed through a mechanism reminiscent of Mimblewimble transactions [9,12]. While the main innovation of Mimblewimble was the compact representation of unspent transaction outputs, their ephemeral quality is the quintessential base for the claim to privacy. In a post-quantum setting, an increase in data volume seems unavoidable due to the large public keys, signatures, and zero-knowledge proofs. However, Promise transactions have the same ephemerality, and therefore the same claim to privacy, as Mimblewimble transactions do. Moreover, in contrast to Mimblewimble, Promise transactions are non-interactive and support scripts.

Promise combines the privacy-preserving transactions with the reputation system of the Pledge into an identity system whereby identities are identifiable with public keys. If the Pledge contract stipulates it, repayments must bear a signature under an authorization public key, which may or may not be the same as the public key of the Borrower. Authorized Parties can make repayments in the Borrower's name without access to his secret key material. This is accomplished through a *proxy re-signature scheme*, whereby Bob can transform his own signatures into signatures that are indistinguishable from authentic signatures produced by Alice, provided that Alice relays the proxy information to enable this transformation. This level of refined access control enables a public dissociation by the Lender from unauthorized transactions.

Like the original protocol, the improved version of Promise supports smart contracts based on the Ethereum Virtual Machine and uses the keyblock/microblock distinction of Bitcoin-NG [7]. Finally, a new feature of the improved protocol is a reputation system designed to enable the rewarding of active nonminer nodes in a way that is entirely independent of the consensus mechanism.

2 Data structures

2.1 Private Transactions

Promise transactions are inspired by Mimblewimble transactions. Unfortunately, a direct translation of the cryptography underlying Mimblewimble into post-quantum primitives remains challenging. Nevertheless, it is possible to decompose Mimblewimble transactions into their functional properties, and achieve them separately with post-quantum primitives. These properties are:

- *Confidentiality.* Amounts remain hidden.
- *Ephemerality.* Unspent transaction outputs are not linked to spent transaction outputs except through the transaction that spends them. Spent transaction outputs can be discarded at no loss to security. This operation is known as *cut-through.*
- *Security from theft.* Transactions only happen if the spenders agree to it.
- *No new coins.* No new money is created. Technically, this property is achieved separately by the *zero-sum-proof*, which establishes that the sum of all inputs is equal to the sum of all outputs; and by the *rangeproofs*, which establish for each output that the amount is zero or larger.

In Mimblewimble, the unspent transaction output (UTXO) amount a is encoded by a Pedersen commitment of the form $g^a h^r$ where r is a randomizer known only to the owner of the UTXO, and $g, h \in \mathbb{G}$ are generators of the group $\mathbb{G}$ with unknown relative discrete logarithms. A valid transaction requires that all inputs sum to all outputs; computing this sum homomorphically from the Pedersen commitments produces $h^{r^\star}$ for some $r^\star$ held jointly by the spenders and receivers. Since $h^{r^\star}$ has the format of a Schnorr public key, the spenders and receivers can generate a signature on the transaction that is valid under this public key. This signature doubles as a zero-sum-proof and as a proof of agreement. Additionally, rangeproofs establish that all commitments represent valid amounts. Unfortunately, since the randomness of the receiver's commitment must be known only to him, the spenders and receivers must run an interactive protocol to generate the signature.

Promise transactions achieve all these properties plus non-interactivity. This implies stealth addresses: a user uploads his public key once, and receives any number of payments thereafter, with none of them being linkable to the published public key or to each other. However, ephemerality is a stronger construction because it additionally enables cut-through.

Address Public Key. An address public key is a pair (spk, epk), where:

- *spk* is the public key of a signature scheme with *public key re-randomization*, which is the ability of a third party to transform a given public key into one that looks completely different. However, the holder of the matching secret key can, after obtaining the used randomness, produce signatures under the new public key.

  Public key re-randomization is not a standard property of signature schemes. However, we observe that certain multivariate quadratic (MQ) signature schemes satisfy this property. The public key there is a list of quadratic polynomials $\mathbf{P} \in (\mathbb{F}_q[x_1, \ldots, x_n]_{\leq 2})^m$ which look random despite hiding a secret trapdoor behind linear transforms $T \in \mathsf{GL}_m(\mathbb{F}_q)$ and $S \in \mathsf{GL}_n(\mathbb{F}_q)$, *i.e.*, $\mathbf{P} = T \circ \mathbf{F} \circ S$ for some quadratic map $\mathbf{F} \in (\mathbb{F}_q[x_1, \ldots, x_n]_{\leq 2})^m$ that admits the efficient inversion. The public key is randomized by computing $A \circ \mathbf{P} \circ B$ for some random $A \xleftarrow{\$} \mathsf{GL}_m(\mathbb{F}_q)$ and $B \xleftarrow{\$} \mathsf{GL}_n(\mathbb{F}_q)$. The secret key holder can produce signatures as in the regular case by inverting the sequence of maps $A \circ T$, $\mathbf{F}$, and $S \circ B$. The unlinkability between the re-randomized public key and the original is guaranteed by the post-quantum hardness of the isomorphism of polynomials (IP) problem [11]. Promise uses the Unbalanced Oil and Vinegar (UOV) MQ signature scheme with field lifting [4].
- *epk* is the public key of a public key encryption scheme. Ciphertexts must be unlinkable to the public keys that were used to create them.

  While this property is not standard either, it is satisfied by ElGamal encryption provided that the generator g is considered as part of the public key. Promise uses the noisy ElGamal cryptosystem NewHope [1].

Address public keys are used for receiving payments. In contrast, identity public keys and authorization public keys are used for contract agreement and payment authorization in the context of the Pledge, see Sect. 2.2.

Unspent Transaction Output. An unspent transaction output (UTXO) is a tuple $(amt, mast\_hash, ctxt)$ where:

- *amt* is either an explicit amount, given by an integer greater than zero; or else a cryptographic commitment to such an integer. The commitment scheme must be compatible with the zero-knowledge proof used in transactions. Promise uses the proof system of Bootle *et al.* [5] in combination with the commitment scheme from Applebaum *et al.* [2].
- *mast_hash* is the root hash of a Merkleized abstract syntax tree (MAST) [10]. The script language is identical to Qtum script [6].
- *ctxt* is an encryption, under the receiver's encryption public key, of the randomness used for the re-randomization of the signature public key.

In general, the *mast_hash* field commits to a script that simply verifies a signature on the transaction under a given public key. This public key is exactly the re-randomized public key and it is hardcoded in the script, whereas the signature must be provided as an explicit assignment to the undetermined variable. By decrypting the ciphertext *ctxt*, the owner of the UTXO can obtain the randomness, re-randomized public key, and full script whose hash is *mast_hash*. However, the *mast_hash* can commit to a more complicated script, in which case enough branches of the abstract syntax tree must be provided, along with assignments to the variables occurring therein, in order for the script to validate the transaction. We refer to these opened branches of the abstract syntax tree as *authentication paths*, even if the tree consists of one leaf.

Transaction. A transaction consists of the following elements:

- M input UTXOs and N output UTXOs called the *inputs* and *outputs*.
- M MAST authentication paths, along with assignments to the undetermined variables.
- A zero-knowledge proof showing that:
  - for each output UTXO, the amount is positive;
  - in total, no new coins are created.

Like in Mimblewimble, the input UTXOs into one transaction cancel against the output UTXOs that generated them, and as a result, *cut-through* is possible. The only data needed by verifiers of new transactions is the present state of the UTXO set. Moreover, UTXOs are never reused.

2.2 The Pledge

Promise provides a data structure called *The Pledge*, which is a contract to repay and agreed on between the following parties:

- *Borrower* or *Payer*: A Borrower is a party who enters into a Pledge to receive a one-time initial payment *from The Pledge* and to make a sequence of repayments *to The Pledge* in return for the initial funding.
- *Lender* or *Payee*: A Lender is a party that enters into a Pledge contract to deliver a payment to the Pledge and receive a stream of payments from the Pledge over a period of time. Lenders assume repayment risk.
- *Originator*: An Originator is a party that sources Lenders and Borrowers and assesses the ability of the Borrower to perform the Promise contract. Originators may or may not assume repayment risk.
- *Authorized Payer*: An Authorized Payer is a party who is authorized to make repayments in the Payer's name.

Contract Terms. Technically, a Pledge consists of the following information:

- A list of payment events called the *clauses*; each clause consists of
  - The time interval
  - The amount
  - The *mast_hash* of the destination
  - (Optional) the authorizer public key for the payment
- The identity public keys of all parties to the contract
- As many signatures, each valid under a unique identity public key
- For each unique *mast_hash* appearing in the clauses, a non-negative number representing the associated *outstanding spendable balance.*
  - This state variable is initially set to zero but is updated as incoming payments arrive or outgoing payments are made

Identity and Authorization Public Keys. Identity public keys are used to sign Pledges. Moreover, they can be used to track a party's payment history over time and derive a measure for their credit reputation.

In contrast, authorization public keys are used for access control with respect to incoming payments. If the payment clause refers to an authorization public key, the payment will be invalid unless it is adjoined with a signature that is valid under that public key.

Both types of public keys admit *proxy re-signing*: the capability to transform a valid signature under Alice's public key into a valid signature under Bob's public key, given some public transformation information generated by Bob [3]. Specifically, Promise uses a variant of the scheme of Fan *et al.* [8].

Paying into a Pledge. Any incoming payment into the Pledge is a transaction whose *mast_hash* is identical to the Pledge hash. Additionally, this transaction identifies the Pledge clauses it satisfies; these clauses, in turn, identify the *mast_hash*es of the Payees. If the transaction is valid, the outstanding spendable balances in question are updated by the given amount.

If the clause defines an authorization public key, the transaction must be adjoined with a valid signature under this public key. This enables various levels of refined access control. The authorizer, who holds the matching secret key, controls who can make these payments by choosing to whom to communicate the proxy resigning information. If the authorizer is the Lender, he authorizes payments from any member of a group he decides on. If the authorizer is the Borrower, he can enable third parties to pay in his name. A third party authorizer can mediate between the constraints of the Borrower and Lender. Alternatively, the absence of an authorization public key allows all incoming payments by default.

The possibility to cultivate payment reputation is a key feature of the Promise ecosystem. In order to maintain this reputation, all agreed-upon payments must be made on time, regardless of business circumstances. If that is not possible, a Payer can maintain his reputation by enabling an Authorized Party to pay in his name, possibly in exchange for another Pledge contract to repay this intervention. The use of proxy re-signing for this purpose serves two purposes: First, the intervention remains hidden from the Payee. Second, the third party who agrees to help out can be certain that the money is being spent on the Pledge payment, and not on something else.

Paying out from a Pledge. In order to spend all or part of the outstanding spendable balance, a Payee makes a transaction referencing the Pledge and *mast_hash*, and provides a MAST authentication path along with satisfying assignment to the variables. The transaction is valid only if the input amount is less than or equal to the outstanding spendable balance. Any change is automatically routed back to the outstanding spendable balance.

Alternatively to spending, the owner can modify the *mast hash*. To do this, he issues a *mast hash update* message, which consists of:

- The Pledge identifier
- A new *mast hash*
- A MAST authentication path and satisfying variable assignment for the old *mast hash*

In the event of a *mast hash* update, the old *mast hash* is replaced by the new in every clause that has not yet been satisfied. The outstanding spendable balance is transferred to the new *mast hash*. Future payments to the old *mast hash* are automatically redirected to the new one. A *mast hash* update thus enables the Payee to transfer spending rights to other parties.

Instead of providing a new *mast hash*, a *mast hash* update can replace the old *mast hash* by a program in the VM language. An incoming Pledge payment will trigger its execution, and it will redirect the funds as stipulated by the code. The code may even stipulate that the incoming payment be used to pay another obligation in another Pledge. This *Pledge chaining* can automate the flow of payments across Pledges and enables users to take new loans against Pledge spending rights while absorbing the repayment risk.

3 Consensus Mechanism

The consensus mechanism of Promise is described in detail in the original whitepaper [15]. We summarize the key points here.

Proof-of-Work. Promise uses a proof-of-work puzzle to establish consensus. Compared to the alternative of proof-of-stake, proof-of-work provides better access to participation: *anyone* can become a miner, not just present coin holders. Also, this choice solves the bootstrapping problem of having to find an equitable initial distribution of coin holders. Promise uses a variant of Dagger-Hashimoto / Ethash [13,14] for the proof-of-work algorithm.

Transaction Output Scaling. Promise uses the transaction throughput scaling techniques proposed by Bitcoin-NG [7]. In essence, Bitcoin-NG makes a distinction between *microblocks*, which are issued by the epoch administrator to collect transactions, transaction proofs and *keyblocks*, which elect an epoch administrator via a proof-of-work puzzle. Malicious epoch administrators may be punished when other nodes discover the fraud and‘ prove it.

Block Size and Time. The post-quantum signature public keys and zero-knowledge proofs used in Promise transactions are large. To account for this size increase, the block size limit must be larger by several orders of magnitude. Therefore, Promise has a microblock maximum size of 16MB, a microblock time of 3 minutes, and 6-7 minutes for the keyblocks.

Difficulty Retargeting. Newly launched coins tend to be the target of intentionally volatile mining hash rates, particularly by large-scale nodes and large-scale mining pools. In order to protect the network from large disruptive movements, Promise sets the difficulty target in a way that protects long-term and smaller-sized nodes. In particular, Promise uses the Linearly Weighted Moving Average (LWMA) #3 algorithm to accomplish this goal [16].

4 Virtual Machine and Smart Contracts

Protocol provides smart contracts that interface with the Promise UTXO transaction type, while running bytecode produced by compilers of high-level scripts on the Ethereum Virtual Machine (EVM). While the EVM assumes account-based transaction types, the Qtum Account Abstraction Layer [6] provides the interface that enables this conjunction.

The EVM comes with the requirement for a gas pricing system, which rewards the miners for running the script. The differences with respect to the Ethereum model take into account the alternate transaction model. In particular, transactions invoking bytecode scripts must additionally specify a return address for unconsumed gas. The Qtum whitepaper provides a comprehensive treatment of the details [6, § 2.4].

5 Promise Nodes

Proof-of-stake networks incentivize nodes to participate and secure the network but reward them based on the size of their holdings instead, of their expenditure of valuable resources. While Promise is a proof-of-work network for the purpose of establishing consensus, it has a built-in mechanism to support and incentivize non-miner node participation. In a nutshell, nodes can be requested to provide a *proof of participation*. By requesting and verifying these proofs from peers, nodes can gauge their peers' participation score as a measure of their honest and active participation.

The network is not guaranteed to converge on a participation score for any subset of nodes. Nevertheless, the reputation mechanism remains useful. In the event of low discrepancy, protocol participants who are driven by various motivations to nurture the ecosystem may choose to reward highly rated nodes for their honest and active behavior. Furthermore, the algorithm by which this node is selected may be transparent and publicly verifiable. Alternatively, a high participation score may be a goal in and of itself, similar to the high share and uptime ratios many anonymous BitTorrent nodes strive for.

5.1 Proof of Participation

The proof-of-participation is realized as an overlay protocol on top of the regular protocol stack. In particular, every node signs (the hash of) every outgoing message under a public key that is available upon request. In addition to that, the node counts the number of outgoing messages and appends both the counter's value and the signature to the message itself.

Every node keeps track of the last X incoming and outgoing messages by storing their associated tuples ($\textit{hash-of-message}$, $\textit{counter}$, $\textit{signature}$, $\textit{timestamp}$) in the order they were received or delivered. Here X is a number, say 1024, whose optimal value is to be determined by experiments or simulations, or may even be chosen by the node itself. Next, the node treats these tuples as the leaves of a Merkle tree and computes the root. Every Y messages (say $Y = 128$) the Merkle tree is recalculated on the new set of X most recent tuples. The node can be requested to respond with its current Merkle root or an authentication path indexed either by its leaf index or by the hash-of-message. If the node fails to respond or responds with an invalid authentication path, then its participation score is decreased. If one node's account of events differs from that of another node, the participation scores of both are decreased. participation scores are increased with regular interactions and successful audits.

Promise provides a proposed table of participation score updates depending on triggering events. Ultimately, however, the nodes can decide for themselves how to appraise each other's participation score. We stress that the reputation mechanism is not a factor in the establishment of consensus. While the reputation system can be gamed, such a gaming will be noticed and ultimately reduces the value of having a high participation score in the first place.

6 Conclusion

Stronger Promises is an upgrade to the original Promise protocol [15] to additionally provide long-term security and privacy while retaining and even expanding on the ability to disintermediate third party (re)payment processors. This update introduces public-key re-randomization as well as zero-knowledge proofs to the Promise network in order to protect users' privacy, and a proxy re-signature to enable fine-grained access control. All introduced cryptography is post-quantum, *i.e.*, conjectured to resist attacks on large scale quantum computers.

References

1. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange - A new hope. In: Holz, T., Savage, S. (eds.) USENIX Security 2016. pp. 327–343. USENIX Association (2016), `https://newhopecrypto.org/`

2. Applebaum, B., Haramaty, N., Ishai, Y., Kushilevitz, E., Vaikuntanathan, V.: Low-complexity cryptographic hash functions. In: Papadimitriou, C.H. (ed.) ITCS 2017. LIPIcs, vol. 67, pp. 7:1–7:31. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017)
3. Ateniese, G., Hohenberger, S.: Proxy re-signatures: new definitions, algorithms, and applications. In: Atluri, V., Meadows, C.A., Juels, A. (eds.) ACM CCS 2005. pp. 310–319. ACM (2005)
4. Beullens, W., Preneel, B.: Field lifting for smaller UOV public keys. In: Patra, A., Smart, N.P. (eds.) INDOCRYPT 2017. LNCS, vol. 10698, pp. 227–246. Springer (2017)
5. Bootle, J., Cerulli, A., Ghadafi, E., Groth, J., Hajiabadi, M., Jakobsen, S.K.: Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part III. LNCS, vol. 10626, pp. 336–365. Springer (2017)
6. Dai, P., Mahi, N., Earls, J., Norta, A.: Smart-contract value-transfer protocols on a distributed mobile application platform. Qtum Foundation, Singapore, 2017 (2017), `https://qtum.org/user/pages/03.tech/01.white-papers/Qtum\%20Whitepaper.pdf`
7. Eyal, I., Efe, A., Gencer, Sirer, E.G., van Renesse, R.: Bitcoin-ng: A scalable blockchain protocol. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) USENIX NSDI '16. pp. 45–59. USENIX Association (2016)
8. Fan, X., Liu, F.: Proxy re-encryption and re-signatures from lattices. IACR Cryptology ePrint Archive **2017**, 456 (2017), `http://eprint.iacr.org/2017/456`
9. Jedusor, T.E.: Mimblewimble. Defunct Tor hidden service. (2016), `https://scalingbitcoin.org/papers/mimblewimble.txt`
10. Lau, J.: Merkelized Abstract Syntax Trees. `https://github.com/bitcoin/bips/blob/master/bip-0114.mediawiki` (2016)
11. Patarin, J.: Hidden fields equations (HFE) and isomorphisms of polynomials (IP): two new families of asymmetric algorithms. In: Maurer, U.M. (ed.) EUROCRYPT '96. LNCS, vol. 1070, pp. 33–48. Springer (1996)
12. Poelstra, A.: Mimblewimble (2016), `https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf`
13. Various authors: Dagger-Hashimoto. `https://github.com/ethereum/wiki/wiki/Dagger-Hashimoto` (2015)
14. Various authors: Ethash. `https://github.com/ethereum/wiki/wiki/Ethash` (2015)
15. Yoshiro, S.: Promise: A decentralized, peer-to-peer, proxy repayment protocol (2018), `https://www.dropbox.com/s/9tltp56dv05lyno/080718_PromiseP2P-026-Final.pdf?dl=0`
16. Zawy12: LWMA difficulty algorithm #3 (2017), `https://github.com/zawy12/difficulty-algorithms/issues/3`