Non-Monotonic Security Protocols and Failures in Financial Intermediation

Fabio Massacci[1], Chan Nam Ngo[1], Daniele Venturi[2], and Julian Williams[3]

[1]Department of Information Engineering and Computer Science, University of Trento, Trento, Italy
[2]Computer Science Department, Sapienza University of Rome, Rome, Italy
[3]Durham University Business School, Durham, UK

February 25, 2018

Abstract

Security Protocols as we know them are *monotonic*: valid security evidence (e.g. commitments, signatures, etc.) accrues over protocol steps performed by honest parties. Once's Alice proved she has an authentication token, got some digital cash, or casted a correct vote, the protocol can move on to validate Bob's evidence. Alice's evidence is never invalidated by honest Bob's actions (as long as she stays honest and is not compromised). Protocol failures only stems from design failures or wrong assumptions (such as Alice's own misbehavior). Security protocol designers can then focus on preventing or detecting misbehavior (e.g. double spending or double voting).

We argue that general financial intermediation (e.g. Market Exchanges) requires us to consider new form of failures where honest Bob's actions can make honest Alice bankrupt *and therefore* invalidate Alice's security credential of good standing. Security protocols must be able to deal with *non-monotonic security* and *new types of failures* that stems from rational behavior of honest agents finding themselves on the wrong side.

This has deep implications for the efficient design of security protocols for general financial intermediation, in particular if we need to guarantee a *proportional burden* of computation to the various parties.

1 Introduction

Thirty years ago, popular security protocols were essentially authentication protocols. A protocol could have various degree of complexity (e.g. Kerberos [18]

vs TLS [7] vs IKE [9]) but essentially two parties tried to authenticate each other (possibly with the help of a trusted third one). There was no question that an honest party could invalidate the evidence of *the* other honest party and there was no issue of computational load because i) each party's main goal was actually receiving the other party security evidence and ii) they participated equally to the protocol[1].

In the past decade, with the emergence and practical deployment of multi-party-computation the *number of parties* participating to a protocol has massively increased[2]. These parties do not talk to each other, they talk to the ensemble *and* some parties might be far more active than others. Yet they potentially share the same burden in computational effort in generic MPC.

At this point there are some interesting questions to make:

- Is security evidence always monotonic as the number of honest parties increases? What type of failures can materialize if this is not the case?
- If some application requires non-monotonic security are there design implications if some parties are more active than others?

As an example *e-cash or voting protocols are essentially monotonic* in terms of legitimacy of digital assets: valid security evidence (e.g. commitments, blinded signatures, etc.) accrues over protocol steps performed by honest parties. Alice's security evidence for a correctly casted vote is not impacted by Bob's correctly casted vote, no matter how many Bobs join, and what they votes. Bobs may tip the balance of the election but not make Alice's vote invalid. This monotonic accumuation of the security legitimacy of digital assets is visible in the security proofs for cash fungibility in ZeroCash [3], or vote's eligibility in E2E [10]. We illustrate this technically in Section 2.

In contrast, *general financial intermediation is not monotonic*: Alice's asset might be proven *cryptographically* valid by Alice (given the current market price her inventory is above her debts) and later made *economically* invalid by the honest Bob who, by just offering to sell assets and thus changing the market price, can make Alice bankrupt without any action on her side. We illustrate this technically in Section 3.

This means that *new type of failures* are possible: *honest failures and failures by omission*. The former, are managed by Exchanges in the current centralized intermediation. The Exchange makes sure Alice deposits enough cash in advance, may eventually suspend trading and eventually absorb Alice's honest losses (acting as a sort of insurance). In a distributed system implementing financial intermediation nobody can absorb Alice's 'honest losses'. Hence the

[1] Obviously the server would have had more load than a client, but this only happens because the server participates to several authentications with several clients at once.

[2] The largest claimed example is the Danish sugar beet auction where 1229 Danish farmers auctioned their production [4]. However, an actual technical reading of the paper reveals that there were only three servers performing MPC over the secret shares generated by the 1200 bidders. As we will illustrate in Section 3 it is actually a good example of a monotonic security protocol.

protocol might need to consider mechanisms to manage this (not unlikely) possibility.

Failures by omission are more critical, especially for non-monotonic protocols. In the example above, as no one but Alice can prove the validity of her standing, whenever a new order arrives and changes the market price she has to publish some (cryptographic) proofs for her valid inventory[3]. If she discovers that her inventory is not valid and learns that she cannot benefit from participating in the protocol anymore, she would simply stop joining the step and any multi-party-computation protocol in the all-but-one security model hang waiting for her messages.

One might argue that a well designed protocol might *fail safe* so that nothing is disclosed and the parties could restart as if nothing happend. From the perspective of the other traders this would rather be *fail uselessly* as there cannot be a market if you can walk abruptly away as soon as you are unhappy with the likely outcome, irrespectively of what you promised to do.

2 Monotonic Security behavior

To define monotonicity of security credentials we focus on a single legitimate protocol run that comprises of multiple steps (and potentially never stops)[4]. Clearly, the security evidence in a step must be valid immediately after the step completed. In the next steps, other *honest* parties may perform some actions. If such actions do not invalidate the security evidence, a protocol is monotonic. Otherwise it is non-monotonic.

For example, a single run of ZeroCash [3] starts with initiating a genesis block and continues to expand the chain to include transactions (The protocol never stops). To make a transaction, a payer simply broadcast the payment information (encrypted and its correctness is proven in zero-knowledge) and the miners find a Proof-of-Work [19] to converge on the transaction result. To do so, the miners first must verify the payment information's correctness by checking the zero-knowledge proofs provided by the payer: (i) the spending coin belongs to an unspent set of coins maintained as a Merkle Tree [17]; (ii) the payer knows a secret parameter (ρ) to unlock the aforementioned coin; and (iii) the transfer amount is well within availability. Another transaction (except for double-spending in which the same coin is paid twice), which would claim another coin, cannot invalidate any of the above proofs.

Similarly, in E2E voting [10], a voter will receive from the Election Authority a vote card giving the voter an authentication code and a vote code. A vote is only valid with the correct authentication code and well-formed vote code. Hence its eligibility (the authentication code and the vote code's correctness) cannot be changed by another vote which claims another authentication code

[3] See an additional discussion in [16] and a concrete implementation in [15].

[4] Security evidence created during a protocol run should not extend beyond the protocol run. Several protocol failures are indeed due to protocol design errors where a credential could be used across sessions [1].

(again, except for double-voting in which the authentication code is used twice) as the other votes yield no direct effect against such vote.

The same phenomenon happens for privacy-preserving reputation systems (e.g. [21]) which evaluate information quality and filter spam by providing linkage between user actions and feedback. In such systems, Alice's reputation, once gained, cannot be affected by Bob's actions to gain his own reputation. Hence security evidence for reputation grows monotonically over honest traders actions.

3 Security of Financial Intermediation is non-monotonic

A financial intermediation service provider, which acts as a counter-party for other participating parties, can be as simple as an auction house, a voting system or a reputation system to be as complex as a futures exchange. Table 1 summarizes the monotonic versus non-monotonic steps in the example of an auction house and a futures exchange [2].

Table 1: Monotonicity vs Non-monotonicity

Auction House		
Steps	**Mono**	**Non-mono**
Authenticate a bidder (optional)	X	
A bidder makes a bid	X	
A bidder proves to have money	X	
Determine winner (in multiple rounds)		X
Voting System		
Steps	**Mono**	**Non-mono**
Authenticate voter	X	
Authenticate vote	X	
Determine winner	X	X
Reputation System		
Steps	**Mono**	**Non-mono**
Accrue reputation	X	
Prove reputation	X	
Futures Exchange		
Steps	**Mono**	**Non-mono**
Authenticate a trader	X	
Make a quote (in a round)	X	
Prove valid order	X	
Prove inventory validity		X
Match a trade	X	
Mark to Market		X

In the first scenario, multiple bidders join the auction with a pre-defined bal-

ance. Each bidder will take turn to make bids by raising the highest price until the winner is identified (as the one who bids the highest). Security requirement in an auction house only includes checking the balance of the bidder's account to be able to satisfy the bids (authentication is only optional, an anonymous auction requires no identity to make a bid). There is no other way to change the validity of a bid once it has been proven except when the owner changes the price to be higher than the available cash in a new bid. Hence this security requirement is monotonic.

To determine the highest bid by bidding against a fixed price is mostly monotonic as shown in this case (only the proclaim winner step needs to be non-monotonic as the winner can change after each bid, hence Alice's proof to be the owner of the highest bid can be invalidated by Bob once he makes a bid that is higher in the next round).

The famous Danish Sugar Beet auction [4] was actually an example of a monotonic bidding against fixed prices. There were 400 fixed price levels and everybody bidded the amount of product they would like to buy (or sell) at each price level. Bob's bid (cryptographically represented as three secret shares) would not make Alice's bid invalid (which were three other independent shares). The three servers (each receiving one share by each bidder) would then perform a MPC computation to add up the quantities at each price level and determine the mid price (where supply would equal demand). Everybody who had bid at that price would actually have to sell/buy.

Similarly, in an e-voting system (e.g. [10]), to authenticate a voter and a vote is monotonic while determine winner *might be* non-monotonic if the result is determined by multiple rounds of voting. A reputation system (e.g. [21]) is fully monotonic.

Differently, in a futures exchange [2], such as Chicago Mercantile Exchange, multiple traders participate with an initial margin to trade futures contracts, a standardised legal agreement between two parties to buy or sell an underlying asset at specified price agreed upon today with the settlement occurring at a future date [20]. Traders take positions (accumulating contracts in inventory) by posting buy and sell orders which effectively changes the market price and directly affects the validity of all trading inventories[5]. Thus the security requirement now involves all parties after an action made by a party. Once an order has been proven valid by a trader, other traders have to come in and prove their inventory valid regarding the new order as their old proofs are discarded when the market price changes. As a result, the security requirement for a futures exchange is non-monotonic: an action made by a trader upon changing the market price immediately invalidates (economically) all validity proofs of other traders[6].

A futures exchange is a good example of a non-monotonic protocol. In the next sections we will discuss the non-monotonic behavior's effect against the new failures and design implications of such non-monotonic protocol.

[5] A formal definition of a Futures Market is given in [16] (Section IV)

[6] See an additional discussion of non-monotonic security in [15] (Section V, Remark 1)

4 Design Implication of Non-Monotinicity: the "Proportional Burden" of Computation

All security protocols implicitly satisfy a *Proportional Burden:* Each computation should be mainly a burden for the party benefiting from it (e.g. Alice is expected to do more work to cast her vote than when Bob is casting his vote). Other parties should join the protocol only to avoid risks (e.g. failed solvency or protect anonymity).

This is a *practical* constraint, and not a *security* one. Such constraint is immaterial in classical security protocols such as user authentication (every user gets the token he ask for), or multi party computation applications such as auctions where everybody makes one bid, or e-voting when everybody casts one vote.

This is definitely not true for security protocols implementing general financial intermediation. In a stock market, most "retail traders" make few quotes, but "algorithmic traders" (typically speculators) make and cancel *thousands* of quotes.

For example, the empirical study in [14]. showed that retail and institutional investors are 71% of traders in the TSX market but only make 18% of the orders. Traders responsible for the bulk of the over 300K orders per day were algorithmic traders who, in 99% of the cases, only submitted limit orders that would never be matched in an actual trade. Any implementation should reflect this practical constraint. Indeed, Centralized Exchanges charges differently based on the number of quotes.

The same thing happens in the Bitcoin network. A transaction from a payer to a payee has to leave some (small) amount to be collected as transaction fee. A miner in the Bitcoin network is compensated for their effort with those transaction fees upon finding the Proof-of-Work to extend the longest chain [19]. As a result, the more transactions a payer makes the more fees he has to pay to the miners.

Monotonic security allows efficient optimizations [21] as a costly multi-party computation (MPC) with n interacting parties may be replaced by n independent (possibly zero-knowledge) non-interactive proofs or secret shares.

The sugar beet auction did exactly that: instead of having 1200 bidders performing an MPC operation all together it had only three servers doing MPC. Each bidders actually submitted only three secret shares to the three servers. Monotonicity of the underlying financial model made it possible to implement it with a monotonic security protocol.

This replacement is also possible when a party only need to make changes to their old secret values based on some public information and prove the correctness in zero-knowledge. This happens in ZeroCash transaction's correctness [3]. It makes it possible for a party to stay off-line and only connect on demand as well as allowing public verification (the proof of payment in ZeroCash can be verified by any party, even the newly arrived ones).

In the general case, the financial intermediation system corresponds to a

security reactive functionality [5] and changes its internal state because an agent performs a valid move which updates the public information and her own private information. If an agent's legit move can unpredictably make another agent's state invalid the system as a whole as a whole must transit to a new state where the legit move is accepted and the invalid state is fixed. This is intrinsically not monotonic as the arrival of one security credential might make economically invalid all the other security proofs cumulated so far.

The solution would be to implement the whole functionality as MPC. Let alone any efficiency consideration[7], this would be unacceptable given the large variance in trading efforts: some traders only make few operations, others can make gazillions of them. In the cited example [14], it is hard to believe that retail investors would be willing to pay CPU and network resources so that speculators could securily and anonimously make their 245.000 vacuous bids against the actual 5000 trades.

A solution would be to require each trader to prove the constraint satisfaction of the economic validity of the order again when new order arrives. However this conflicts with the market's anonymity requirement in case *only one* party cannot prove the validity. This leads to dangerous seconomic vulnerabilities such as price discrimination attack [16].

Hence, the challenging part of the protocol construction is to identify the minimal core of the state of the reactive security functionality implementing the financial intermediation service provider that would account for its non-monotonic behavior in the legitimacy of traders and assets[8]. This is the only part where MPC needs to be used.[9]. As shown in [15], this approach can reduce the total burden of computation by retail traders by several orders of magnitude heavier comparing to the generic MPC implementation.

5 Design Implications of Non-monotonicity: Failures by omission

From a security perspective, the above design is only secure-with-abort as an adversary can abort the protocol by simply not participating in a joint MPC step. The protocol *fails by omission*. It is true that from a security perspective one can design the protocol to be fail-safe [8], but this is hardly acceptable in practice. Which speculator would join the TSX market mentioned above if any retail investor disconnecting its computer could fail safe to nothing happend and thus avoid being thoroughly shaved? Would institutional or retail investors ever join if any glitch by mistake or mischief by an algorithmic trader could fail safe to nothing done a day of costly MPC computation?

[7]The 1229 parties full MPC variant is still out of reach for the foreseable future as experimental papers typically reported MPC with less than 10 parties [6].

[8]See Section VII of [15].

[9]This does not violate the proportional burden requirement as each trader has the responsibility to prove the solvency if s/he still wants to be in the game

A preliminary observation is that in practice one cannot initialize a market with a self-claimed account. The cash that get deposited into the market must be backed by a verifiable source where a debit is acknowledged by every market participants, e.g. ZeroCash. Hence, such source must be able to publicly verify the validity of the transactions resulting from the market's operation at the end of the day to credit each the account with the corresponding amount.

An approach to penalize a faulty participant upon aborting in an MPC with digital cash is to make the adversary lose some digital cash. The works in [12] and [13] require the adversary to make deposits and forfeit them upon dropping out. Technically the parties have to stake *increasing deposit in a fixed order* since order of revelation is important (the see-saw mechanism, [12, p. 7]) for the aforementioned penalty mechanism to work[10] To participate in a game where x is at stake, the first trader completing the protocol deposits $n \cdot x$, the second trader deposits $(n-1) \cdot x$, and the n-th trader deposits x.

Unfortunately those protocols are not usable in any practical scenario when deposits are actually meaningful (i.e. x is truly money and not a LaTeX symbol) as they are *economically unfair*. This is due to the difference in financial capability of traders. Consider a real futures market: the single smallest contract has a value of 1 million (real) dollars. In a low-frequency market (lean-hog futures) there are only few tens of traders but still the trader completing first would have to deposit assets 35x times the stake of the trader completing last, and in large markets more that 500 times larger. It is true that this money would be returned at the end of the protocol, yet while the protocol is in execution the first completing trader would have to borrow 500million dollars for its deposit to make an order worth 1 million...

A better solution against omission is the mechanism of Hawk [11, Appendix G, §B] in which private deposits are frozen and the identified aborting parties cannot claim the deposits back in the withdraw phase. This requires that the protocol must be able to provide security tokens of successful completion and provide identifying evidence not only in case of misbehavior but also in case of aborts. We refer the reader to Section X of [15] for additional discussion.

6 Conclusions

In this paper we have argued that the increasing number of (honest) parties that participate to security protocols makes it possible to distinguish between monotonic and non-monotonic security protocols.

Non-monotonic security implies novel failure modes and novel design challenges for protocl designers. Yet, we have also shown that some of them could actually be addressed.

[10] In some cases this fixed order might interfere with the security goal, if the order of actions may leak some information on who started the process.

References

[1] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. In *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*, pages 122–136. IEEE, 1994.

[2] F. Allen and A. M. Santomero. The theory of financial intermediation. 21(11-12):1461 – 1485, 1997.

[3] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. pages 459–474, 2014.

[4] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. P. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, et al. Secure multiparty computation goes live. In *Financial Cryptography*, volume 5628, pages 325–343. Springer, 2009.

[5] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. pages 494–503, 2002.

[6] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. pages 1–18, 2013.

[7] T. Dierks and C. Allen. The tls protocol version 1.0. 1999.

[8] L. Gong. *Fail-stop protocols: An approach to designing secure protocols.* 1994.

[9] D. Harkins and D. Carrel. The internet key exchange (ike). Technical report, 1998.

[10] A. Kiayias, T. Zacharias, and B. Zhang. An efficient e2e verifiable e-voting system without setup assumptions. 2016.

[11] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. pages 839–858, 2016.

[12] R. Kumaresan, T. Moran, and I. Bentov. How to use bitcoin to play decentralized poker. pages 195–206, 2015.

[13] R. Kumaresan, V. Vaikuntanathan, and P. N. Vasudevan. Improvements to secure computation with penalties. pages 406–417, 2016.

[14] K. Malinova, A. Park, and R. Riordan. Do retail traders suffer from high frequency traders. *Available at SSRN*, 2183806, 2013.

[15] F. Massacci, C. N. Ngo, J. Nie, D. Venturi, and J. Williams. Futuresmex: Secure, distributed futures market exchange. In *2018 IEEE Symposium on Security and Privacy (SP)*, volume 00, pages 453–471.

[16] F. Massacci, C. N. Ngo, J. Nie, D. Venturi, and J. Williams. The seconomics (security-economics) vulnerabilities of decentralized autonomous organizations. In *Proceedings of the twenty-fifth Security Protocols Workshop*, 2017.

[17] R. C. Merkle. A digital signature based on a conventional encryption function. pages 369–378, 1987.

[18] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer. Kerberos authentication and authorization system. In *In Project Athena Technical Plan*. Citeseer, 1987.

[19] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[20] D. F. Spulber. Market microstructure and intermediation. 10(3):135–152, 1996.

[21] E. Zhai, D. I. Wolinsky, R. Chen, E. Syta, C. Teng, and B. Ford. Anonrep: Towards tracking-resistant anonymous reputation. pages 583–596, 2016.