On the (Limited) Power of Non-Equivocation

Allen Clement
MPI-SWS
aclement@mpi-sws.org

Flavio Junqueira
Yahoo! Research
fpj@yahoo-inc.com

Aniket Kate
MPI-SWS
aniket@mpi-sws.org

Rodrigo Rodrigues
CITI / Universidade Nova de Lisboa
rodrigo.rodigues@fct.unl.pt

ABSTRACT

In recent years, there have been a few proposals to add a small amount of trusted hardware at each replica in a Byzantine fault tolerant system to cut back replication factors. These trusted components eliminate the ability for a Byzantine node to perform equivocation, which intuitively means making conflicting statements to different processes.

In this paper, we define non-equivocation and study its power in the context of distributed protocols that assume a Byzantine fault model. We show that non-equivocation alone does not allow for reducing the number of processes required to reach agreement in the presence of Byzantine faults in the asynchronous communication model, by proving a lower bound of $n > 3f$ processes for agreement with non-equivocation. However, when we add the ability to guarantee the transferable authentication of network messages (e.g., using digital signatures), we show that it is possible to use non-equivocation to transform any protocol that works under the crash fault model into a protocol that tolerates Byzantine faults, without requiring an increase in the number of processes.

Categories and Subject Descriptors

B.8.1 [**PERFORMANCE AND RELIABILITY**]: Reliability, Testing, and Fault-Tolerance; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; D.4.6 [**OPERATING SYSTEMS**]: Security and Protection—*Cryptographic controls*

General Terms

Algorithms, Reliability, Security, Theory

Keywords

Non-equivocation, Transferable authentication, Trusted hardware

PODC'12, July 16–18, 2012, Madeira, Portugal.

1. INTRODUCTION

The Byzantine fault model [15] is an important foundation for building protocols that offer strong guarantees in the presence of processes that fail in ways other than silently crashing. This model has been adopted by a series of practical replicated systems [5] that are resilient to arbitrary faults, which can originate from issues ranging from software bugs to malicious attacks.

In recent years, several proposals [7, 16] have emerged to add a trusted hardware component to each process running in a Byzantine fault tolerant replicated system in order to make the replication protocols more efficient, most notably by reducing the number of processes that are required to build a replicated state machine. The capability behind this trusted component is to prevent a Byzantine process from equivocating other nodes, which intuitively means making conflicting statements to two or more other processes. Since this can be enforced using a very simple design that can be implemented on today's hardware, the expectation is that this new capability might be widely adopted in the design of more efficient secure distributed systems.

In this paper we set to study the power of non-equivocation. To this end, we formally define non-equivocation in the context of a distributed system with Byzantine processes, and we study whether non-equivocation can aid in finding efficient solutions to the problems of agreement and reliable broadcast.

Our first finding is that non-equivocation is unable to reduce the number of processes required to solve Byzantine agreement in an asynchronous system. In particular, we prove a lower bound of $3f + 1$ processes to solve agreement in a system model that considers both Byzantine faults and non-equivocation. This proof does not contradict the benefits in terms of reducing replication factors achieved by systems like A2M [7] and TrInc [16], since the proof assumes the nonexistence of a primitive that allows for the transferable authentication of network messages (e.g., digital signatures). The second result in this paper focuses precisely on the system model where the unforgeability of network messages can be guaranteed. In this setting, we show how we can use non-equivocation to obtain a generic transformation from a protocol that works under the crash fault model into a protocol that provides the same guarantees under the Byzantine fault model, without requiring an increase in the number of processes. Such transformation can be used, for instance, to solve consensus with $2f + 1$ processes.

With this work, and namely by introducing a new system

model where Byzantine processes are constrained in their behavior by the presence of trusted hardware, we hope to initiate the formal study of the role of trusted computing in the design of secure distributed protocols, an area that is gaining increasing visibility in the security community. Furthermore, we envision that our generic transformation can be an initial step in a research avenue of building distributed protocols that are secure by design, at the cost of adding inexpensive trusted hardware at each process plus a modest increase in protocol complexity.

1.1 Related Work

Chun et al. [7] observe that one of the most disruptive behaviors of Byzantine processes is lying in different ways to different clients or servers. They coin the term *equivocation* for this adversarial behavior, and propose a small trusted computing facility, attested append-only memory (A2M), that makes protocol designs immune to equivocation and enables a lower degree of replication. In particular, using the non-equivocation mechanism from A2M, they provide a Byzantine fault-tolerant state machine replication system that requires only $2f + 1$ replicas instead of $3f + 1$ required in the general Byzantine environment [5]. Levin et. al. [16] reduce the hardware trust assumption required for non-equivocation to a trusted non-decreasing counter and a key that provides unique, once-in-a-lifetime attestations. Nevertheless, the general idea that these papers convey is that using a non-equivocation mechanism results in an improved resiliency bound, which we show not to be true, at least from using non-equivocation *per se*. We instead find that, along with non-equivocation, the use of digital signatures as transferable authentication in A2M and TrInc results in an improved resiliency bound.

The concept of transforming a protocol tolerating f crashes to a protocol tolerating f Byzantine faults with translations using more processes is not new. Bracha [4] defined such a translation for a crash-tolerant consensus protocol to tolerate the same number of Byzantine faults. Coan [8] generalizes this translation to a larger variety of consensus and related protocols. Ho, Dolev, and van Renesse [13] observe that Coan's generalized translation is suitable only for consensus and related problems, and does not work for arbitrary distributed protocols. They instead define the concept of ordered authenticated reliable broadcast (OARcast) over FIFO communication links, and use an OARcast protocol to convert a crash-tolerant system into a system tolerating the same number of Byzantine faults. In follow-up work [14], they further improve their translation to be more scalable and reconfigurable and prove the practicality of the translation by applying it to a link-based routing protocol and an overlay multicast protocol.

Our generic translation provides significant improvement over the above discussed translations: using non-equivocation and transferable authentication mechanisms, we translate a crash-tolerant protocol to a Byzantine fault tolerant protocol without asking for any additional processes or any additional messages (though substantially increasing the message size); we do not restrict our links to be FIFO; finally, our translation is simple, and easy to analyze and specialize for various distributed protocols.

Furthermore, small trusted hardware components have been considered as part of hybrid fault models [18]. Such devices enable a smaller number of replicas and reduce the communication complexity as shown for distributed cryptographic protocols in the security literature [2, 11]. Although these systems inherently use the concepts of non-equivocation and transferable authentication, they do not formally define or study the power of non-equivocation and transferable authentication.

2. SYSTEM MODEL

This section presents our system model, and precisely defines the capability of non-equivocation.

2.1 Fault and communication model

The system consists of n processes $p_1, \ldots, p_n$ connected by a network such that processes are pairwise connected by an asynchronous channel, *i.e.*, messages between two processes can be arbitrarily delayed, or reordered. However, we assume messages are eventually delivered. At most f of n processes may exhibit *faulty* behavior. The remaining processes are *correct*. We consider in this work two distinct models for faulty processes:

Crash. A faulty process does not execute further steps once it crashes;

Byzantine. A faulty process may deviate arbitrarily from the correct behavior of the protocol it implements.

2.2 Computation Model

A system S consists of a complete graph of system processes and an algorithm for each process in the graph. An algorithm for a process is a (possibly infinite state) automaton, which has an initial state Σ_0 and performs the following actions in each step it takes:

- it receives a message through the incoming channels;
- given its state and the message received, it moves to a new state and sends a (possibly empty) set of messages through the outgoing channels.

This allows us to define a *process behavior* $\mathcal{B}_{p,h}$ for process p as a sequence of the form: $\Sigma_{p,0}, (\mathcal{R}_{p,1}, \Sigma_{p,1}, \mathcal{S}_{p,1}), (\mathcal{R}_{p,2}, \Sigma_{p,2}, \mathcal{S}_{p,2}), \ldots, (\mathcal{R}_{p,h}, \Sigma_{p,h}, \mathcal{S}_{p,h})$ such that

- $\Sigma_{p,0}$ is an initial state of the process,
- $\mathcal{R}_{p,j}$ is the j^{th} set of messages received by p[1],
- $\Sigma_{p,j}$ is the state of the process p after processing the message $\mathcal{R}_{p,j}$ while in state $\Sigma_{p,j-1}$, and
- $\mathcal{S}_{p,j}$ is the set of messages (if any) sent by p after entering in the state $\Sigma_{p,j}$.

2.3 Authentication

An authenticated message m is accompanied by an authentication token σ_{p_i} that allows a recipient p_j to verify that p_i generated the message using *verify*(m, σ_{p_i}). A key property of message authentication is that authentication tokens are *unforgeable*: if p_i is non-faulty then *verify*(m, σ_{p_i}) evaluates to true if and only if p_i generated the message m; for a faulty p_i, *verify*$(m, \sigma_{p_i}) \wedge$ *verify*$(m', \sigma_{p_i}) \Rightarrow m = m'$. An authentication token provides *transferable authentication* if correct processes p_j and p_k always evaluate *verify*(m, σ_{p_i})

[1]In some cases $\mathcal{R}_{p,j}$ will be a singleton set.

in the same way, when p_k receives message m and authentication token σ_{p_i} from p_j. An authentication token provides *non-transferable authentication* if process p_j can evaluate *verify*(m, σ_{p_i}) to true while process p_k may not be able to correctly evaluate it. For example, digital signatures provide transferable authentication while arrays of MACs provide non-transferable authentication [1].

2.4 Non-equivocation

A process p equivocates if it sends different messages to different replicas in the same round while it was supposed to send the same message according to the protocol. To prevent equivocation, the system model provides the protocol designer with the capability of *non-equivocation*. This capability consists of being able to validate a pair (k, m) from a process p, where key $k \in \mathbb{N}$ and m is an arbitrary message, with the guarantee that the same key k cannot be used to validate contradicting messages from process p. Let $valid_p(k, m)$ be a predicate that evaluates to true if and only if k validates m and (k, m) has been generated by p. Non-equivocation guarantees that $valid_p(k, m) \wedge valid_p(k, m') \Rightarrow m = m'$. Note that the *valid* predicate is generally realized using the *verify* predicate of the authentication mechanism discussed in Section 2.3.

3. LOWER BOUNDS

In this section, we study the lower bounds for the reliable broadcast primitive using the non-equivocation Byzantine model considered in A2M [7] and TrInc [16]. Contrary to the belief, subjacent to these papers, that the non-equivocation restriction on a Byzantine adversary improves the degree of replication from $n \geq 3f + 1$ to $n \geq 2f + 1$, we observe that signatures (or transferable authentication) also play an instrumental role along with the non-equivocation restriction in improving the resiliency bound. In particular, we prove that assuming at most f Byzantine faults in an asynchronous system of n processes, the non-equivocation restriction and transferable-authentication tokens together, but not individually, are sufficient to improve the resiliency bound to $n \geq 2f + 1$.

Note that our resiliency results also apply to other related distributed computing primitives such as consensus, state machine replication and verifiable secret sharing, since they are stronger problems compared to the reliable broadcast problem we target [6].

3.1 Definition

We start by defining the Reliable Broadcast (r-broadcast) primitive.

r-broadcast is a fundamental primitive for synchronization among a group of processes, which can be characterized by the following *liveness* and *safety* properties.

DEFINITION 1. *In an asynchronous system of n processes having a distinguished sender s and an f-limited adversary with point-to-point authenticated links, a* **reliable broadcast (r-broadcast) protocol** *satisfies the following liveness and safety conditions:*

Liveness.

Sender Termination (L1). *If the sender is correct and sends m, then the sender delivers m.*

Completeness (L2). *If a correct process delivers a message, then all correct processes deliver a message.*

Safety.

Validity (S1). *If the sender is correct and a correct process delivers a message m, then the sender must have sent m.*

Agreement (S2). *If a correct node p_i delivers m_i and another correct node p_j delivers m_j, then $m_i = m_j$.*

Integrity (S3). *A correct node delivers at most one message.*

Note that the r-broadcast primitive cannot guarantee termination of a protocol instance for a faulty sender. It follows from the impossibility of guaranteeing consensus in the asynchronous setting with even a single crash fault [10].

The r-broadcast primitive only needs $n \geq f+1$ in the crash fault model [12] while it requires $n \geq 3f+1$ in the Byzantine model [3]. We include the corresponding resilience-optimal r-broadcast protocols in Appendix A.

3.2 Analysis

We now show two lower bound claims for the degree of replication of r-broadcast when considering non-equivocation and transferable authentication. The first claim assumes that faulty processes cannot equivocate, but there is no transferable authentication. The second claim makes the opposite assumption: the model does not give access to non-equivocation, but does allow for transferrable authentication.

In the following proofs, we use the notion of locality. If two isomorphic subsystems consisting of correct processes are such that corresponding processes start in the same initial states and corresponding input edges to the subsystems carry the same messages, then the two subsystems exhibit identical behavior, independent of the overall systems. This property is called the *locality axiom* [9].

CLAIM 3.1. *Suppose an asynchronous system of n processes such that at most f processes fail under the Byzantine model and $n < 3f + 1$. If processes do not implement transferable authentication and faulty processes cannot equivocate, then there is no solution for r-broadcast.*

PROOF. Proof by contradiction. Suppose that there exists a protocol that solves r-broadcast with $n < 3f+1$ without using transferable authentication even when the adversary cannot equivocate. Without loss of generality, we assume that $n = 3f$.

Suppose that we partition the set of processes n into disjoint subsets A, B, and C, such that each subset contains exactly f processes. Using this partition of the set of processes, we construct several runs of the protocol, the last of which violates a safety property. In all runs there is a distinguished process $s \in A$ that broadcasts a message.

Run 1. Processes in C crash at the beginning of the run (time t_0):

- Process s broadcasts m', all processes in C crash at time t_0, and all other processes are correct;

- Messages between A and B are delivered in a timely fashion;
- By the Liveness and Agreement properties of r-broadcast, all processes in A and B deliver m' by time t.

Run 2. Processes in A crash after processes in B have delivered m', and messages to and from C are arbitrarily delayed:

- Process s broadcasts m', all processes in A crash at time t and all other processes are correct;
- Messages between A and B are delivered in a timely fashion and the steps of processes in A and B are the same as in Run 1 until time t;
- Processes in B deliver m' by time t, since they cannot differentiate Run 2 from Run 1;
- Messages between B and C are delayed until time t', t' arbitrarily large and $t' > t$;
- Messages between A and C are not delivered (processes in A are faulty in this run);
- By the Agreement property of r-broadcast, processes in C deliver m' by time $t'' > t'$.

Run 3. All processes are correct, but messages from and to processes in C are arbitrarily delayed:

- Process s broadcasts m', and all processes are correct;
- Messages between A and B are delivered in a timely fashion and the steps of processes in A and B are the same as in Run 1 until time t;
- Processes in B have all delivered m' by time t, since they can't differentiate Run 3 from Run 2;
- Messages between B and C are delayed until time t';
- Messages between A and C are delayed until time $t''' > t''$.
- Processes in C deliver m' by time t'', since they cannot distinguish Run 3 from Run 2.

Run 4. All processes in B are faulty:

- Process s broadcasts m, processes in B are faulty, and all other processes are correct;
- Messages between processes in B and C are delayed until time t' and the steps of processes in B and C are the same as in Run 3, in which the sender broadcasts m';
- Messages between A and C are delayed until time t''';
- Processes in C deliver m' by time t'', since they cannot distinguish Run 4 from Run 3;
- By Liveness, process s delivers m.

Run 4 constitutes a violation of the Agreement property and contradicts the assumption of a protocol that solves r-broadcast for $n \leq 3f$ processes. □

Note that in the proof of Claim 3.1, Run 4 is valid even if faulty processes cannot equivocate. This is because Byzantine processes in B (in Run 4, Step 2) repeat sequences of messages of another run, while not equivocating. Correct processes in C cannot distinguish between Run 3 and Run 4, since there is no transferable authentication scheme implemented by assumption.

The next theorem considers the opposite setting, where Byzantine nodes can equivocate, but the system model allows for transferable authentication.

CLAIM 3.2. *Suppose an asynchronous system of n processes such that at most f processes fail under the Byzantine model and $n < 3f + 1$. If processes implement transferable authentication and faulty processes can equivocate, then there is no solution for r-broadcast.*

PROOF. Proof by contradiction. Suppose that there exists a protocol such that it solves r-broadcast with $n < 3f+1$ assuming transferable authentication and that faulty processes can equivocate. Without loss of generality, we assume that $n = 3f$.

Suppose that we partition the set of processes n into disjoint subsets A, B, and C, such that each subset contains exactly f processes. Using this partition of the set of processes, we construct three runs. In all runs there is a distinguished process $s \in A$ that broadcasts a message.

Run 1. Processes in B crash:

- Process s broadcasts m and all processes in B crash at the beginning of the run (t_0);
- By the agreement and the liveness properties of r-broadcast all process in A and C eventually deliver m by time t.

Run 2. Processes in C crash:

- Process s broadcasts m' and all processes in C crash at the beginning of the run (t_0);
- By the agreement and the liveness properties of r-broadcast all process in A and B eventually deliver m' by time t'.

Run 3. Processes in A are faulty:

- Processes in A execute the same steps as in Run 1 when exchanging messages with processes in C;
- Processes in A execute the same steps as in Run 2 when exchanging messages with processes in B;
- Messages between B and C are delayed until time $\max\{t, t'\}$;
- Processes in C cannot distinguish Run 3 from Run 1, and they deliver m;
- Processes in B cannot distinguish Run 3 from Run 2, and they deliver m'.

Run 3 constitutes a violation of the Agreement property and contradicts the assumption of a protocol that solves r-broadcast for $n \leq 3f$ processes. □

Note that in the proof of Claim 3.2 the faulty processes in Run 3 do not violate transferable authentication, although the first two steps of the run possibly violate non-equivocation. We also note the similarity between the argument in the proof of the theorem and the argument provided by Pease, Shostak, and Lamport for interactive consistency [17]; we adapted it to the r-broadcast problem.

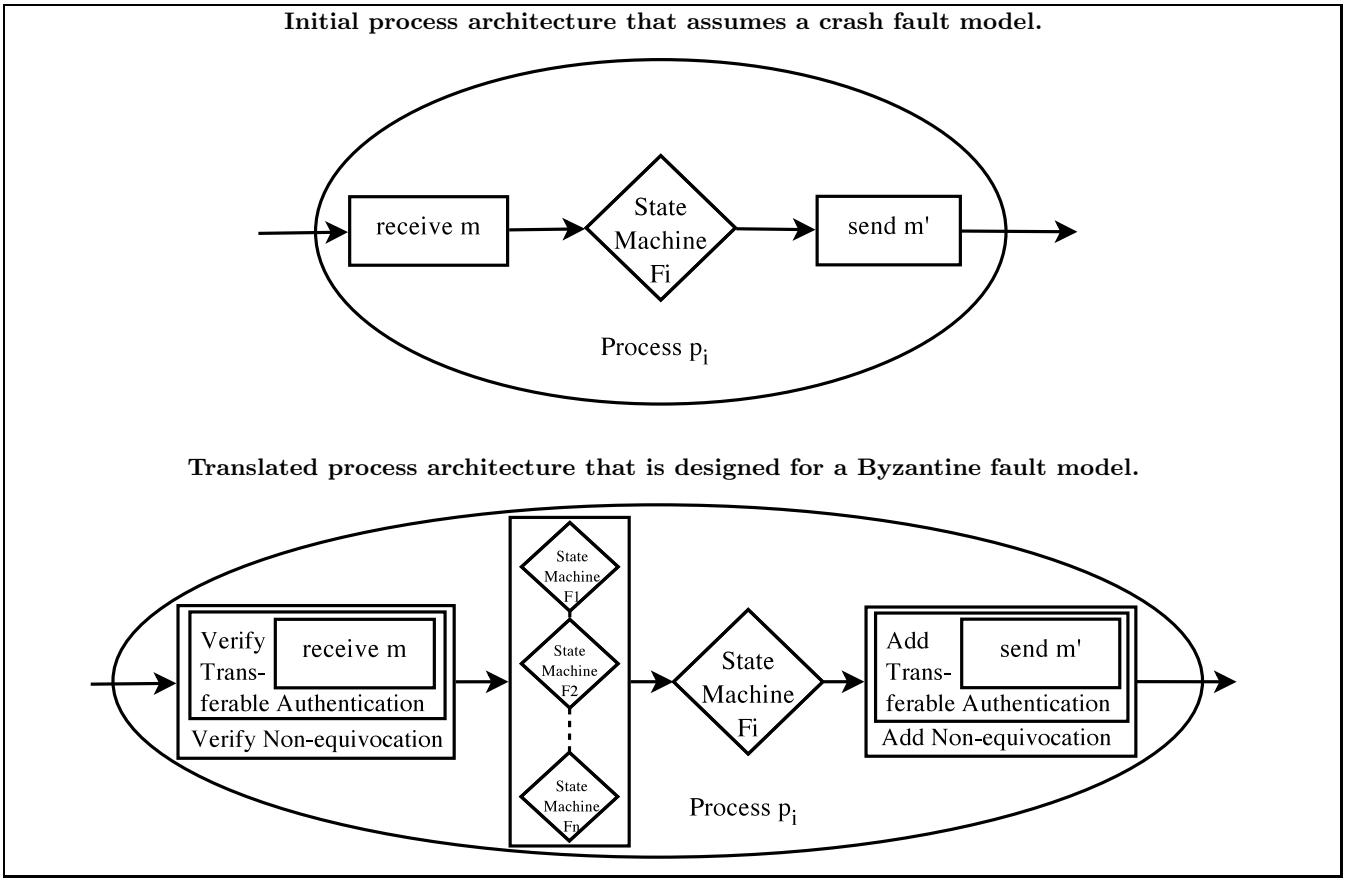

Figure 1: Overview of the translation from a crash fault tolerant system to a Byzantine fault tolerant system.

4. TRANSLATION

In this section we demonstrate the power of combining non-equivocation with transferable authentication. In particular, we show that there exists a translation of a distributed protocol that can tolerate f crash faults into one that tolerates f Byzantine faults using the same number of processes and the same number of (longer) messages. This transformation makes use of both non-equivocation and digital signatures. This result implies that we can solve Byzantine agreement with only $2f + 1$ processes, thus confirming the potential of non-equivocation evidenced by A2M [7] and TrInc [16].

4.1 Requirements

The basic idea of the translation mechanism is to replace the primitives for sending and receiving messages through the network with a more complex protocol that provides the same key guarantees about the contents of the delivered messages that are expected by the original protocol in the crash fault model.

This strategy is similar to the one used by another proposal by Ho *et al.* [13, 14] for transforming protocols that tolerate crash faults into protocols that provide the same guarantees in the Byzantine fault model. In particular, they identify the following key safety properties that are expected by a crash fault tolerant protocol regarding the messages that are delivered by the network, and that must also be provided by our translation mechanism.

Safety. If the sender p_i is correct and a correct process p_j *receives* a message m from p_i, then the sender p_i must have executed *send* with m.

Integrity. If a correct process *receives* a message m, then m is a valid state machine message.

Regarding the Integrity property, message m sent by process p_i is valid if it corresponds to the output of applying a sequence of valid messages to the state machine of the process (and considering that external inputs, like messages coming from clients of a replicated service, are valid), i.e., there is a correct process behavior $B_{p_i,h} = \Sigma_0, (\mathcal{R}_1, \Sigma_1, \mathcal{S}_1), (\mathcal{R}_2, \Sigma_2, \mathcal{S}_2), \ldots, (\mathcal{R}_h, \Sigma_h, \mathcal{S}_h)$ such that $m \in \mathcal{S}_h$ and all $\mathcal{R}_i$ are valid.

Ho *et al.* [13, 14] also include the liveness property that if a correct sender p_i *sends* a message to a correct process p_j, then process p_j will *receive* the message eventually. This property is generally satisfied by asynchronous communication models assumed by many distributed protocols, e.g., the r-broadcast primitive in Section 3.1. However, it is not considered mandatory for every crash fault tolerant protocol, and to make our list of requirements more generic, we do not include this liveness property in the list.

The previous generic transformation work [13, 14] stated another property, which we modify in our set of requirements. The original missing property stated that the transformed network implements a first in, first out (FIFO) policy [13, 14]. However, this is insufficient, since two correct

upon `B-receiving` a message $((k_p, m, \mathcal{R}_p, \mathcal{S}_p), \sigma_p)$ from p:
  verify$((k_p, m, \mathcal{R}_p, \mathcal{S}_p), \sigma_p)$
  Check that $k_p = |\mathcal{R}_p| = |\mathcal{S}_p|$
  For all the messages in $\mathcal{R}_p$ and $\mathcal{S}_p$ check authenticity
  Instantiate copy of state machine in initial state, feed messages inputs in $\mathcal{R}_p$, confirm outputs match those in $\mathcal{S}_p$
  For all the messages in $\mathcal{R}_p$, recursively apply same checks to ensure checks were made by sender
  if all checks pass **then**
    Deliver `T-receive` message m from p
    add $((k_p, m, \mathcal{R}_p, \mathcal{S}_p), \sigma_p)$ to the set of received messages $\mathcal{R}$
  end if

upon receiving a `T-send`(m) to set P from the state machine:
  Increment k
  σ_p = GenerateAuthentication $(k, m, \mathcal{R}, \mathcal{S})$
  `B-send` $((k, m, \mathcal{R}, \mathcal{S}), \sigma_p)$ to all processes in P
  Add $((k, m, \mathcal{R}, \mathcal{S}), \sigma_p)$ to the set of sent messages $\mathcal{S}$

Figure 2: Pseudocode for the generic translation

processes might receive contradictory but valid messages from a Byzantine process (i.e., the final output of two forks, both corresponding to correct executions) without violating FIFO since each correct process only receives one of the two messages. This is a situation that crash fault tolerant protocols do not expect, and may not be ready to handle. We therefore modify this final requirement as follows.

Consistency. If correct processes p_1 and p_2 receive valid messages $m_i \in \mathcal{S}_i$ and $m_j \in \mathcal{S}_j$ respectively from process p_g, then either (a) $B_{p_g,i}$ is a prefix of $B_{p_g,j}$, (b) $B_{p_g,j}$ is a prefix of $B_{p_g,i}$, or (c) $B_{p_g,i} = B_{p_g,j} \wedge \mathcal{S}_i = \mathcal{S}_j$ (where $B_{p_g,x}$ is the process behavior that supports the validity of message m_x).

4.2 Translation Implementation

Now, we show a translation, defined as a set of wrapper functions for processes to send and receive messages, that meets the above listed properties. Given that the terms send and receive will be overloaded, we will make the following distinction throughout our description:

- `B-send` and `B-receive` refer to the lower level sending and receiving of messages through the network in the Byzantine model (i.e., used by the implementation of the translation).
- `T-send` and `T-receive` are the resultant higher level primitives, i.e., the interface exported by the translation layer to the implementation of the original crash fault tolerant protocol sitting above that layer.

The goal of the translation layer is to implement `T-send` and `T-receive` primitives that satisfy the three requirements defined in Section 4.1 despite f Byzantine faults, by leveraging the non-equivocation and transferable authentication mechanisms. Note that our transformation encapsulates and runs instances of the state machine. Consequently, it is not independent of the state machine, and does not clearly separate the communication layer and the application layer.

The safety requirement can be ensured by retrying the `B-send` attempts and by authenticating messages in `B-send` and `B-receive`. Our assumption of transferable authentication is thus sufficient (and even too strong since non-transferable authentication would also suffice).

The integrity requirement requires the receiver to obtain a guarantee that a message from process p_i is based on a sequence of valid messages. In this case, the transferable authentication mechanism helps by allowing p_i to prove the validity of its output message to its receivers by attaching the corresponding signatures (or authentication tokens) of p_i's inputs that led to sending the output message. Then by replaying these authentic inputs, the integrity requirement can be validated.

Finally, the consistency requirement can be enforced through non-equivocation, simply by including a sequence number in the key associated with each outgoing message, thus forcing a total order on the outputs of each process, and by forcing the sender process to also transmit all the previous sequence numbers in each message step.

We therefore reach the translation mechanism that is depicted in a block diagram in Figure 1. In Figure 2 we present the pseudocode for the translation.

The idea is that to `T-send` a message a node must pass it through a non-equivocation layer and a transferable authentication layer. The non-equivocation layer causes the output of the ith state transition that is sent to be associated with key k, and passes the (k, m) message to the next layer. The transferable authentication layer associates with each outgoing message the sequence of signed inputs that led to generating this message, and also signs the message before `B-send`ing it.

On the receiver side, the transferable authentication layer checks that the message is signed by the correct sender, replays the execution of the sequence of inputs encapsulated in the message, checking if the output matches the payload that was received, and recursively applies these checks to every message in the sequence, to ensure that the messages were based on valid inputs. The non-equivocation layer merely has to check that the key k matches the number of inputs in the sequence.

We present a correctness argument for this transformation in Appendix B.

5. CONCLUSION AND FUTURE WORK

Non-equivocation is a practical capability available to the design of systems that tolerate Byzantine faults. It has received some attention from the designers of distributed systems in recent years because of the important practical features it enables: a simple implementation and a lower degree of replication.

In this paper, we formally defined non-equivocation, and showed that, contrary to the belief implied by the initial proposals for using hardware support for non-equivocation, this capability by itself does not decrease the number of replicas required to solve problems like consensus in the Byzantine fault model. However, coupled with transferable authentication, it is possible to reduce the degree of replication. To demonstrate it, we show a generic transformation from a crash-tolerant protocol into one that is Byzantine tolerant, using non-equivocation and transferable authentication, requiring the same number of processes as the crash-tolerant counterpart.

This work opens some near term follow-up research directions, namely it would be interesting to make our translation mechanism more efficient. The current translation uses an expensive mechanism of replaying the execution of almost the entire system to ensure that the message that is received is legitimate. While it shows that a translation exists without increasing the number of processes, it is also clearly not a practical strategy and can be highly optimized, *e.g.*, by reusing previous verifications, and also by using checkpointing mechanisms that allow nodes to agree on a correct view of the system.

Given the potential of non-equivocation to enable more efficient protocols that tolerate Byzantine faults, and the viability of the concept that has been shown by the existence of efficient hardware implementations and its application to existing distributed systems, we believe that the formal study of non-equivocation is an important research direction. Furthermore, the translation we provide is an important initial step towards the design and implementation of distributed systems that are secure by construction, and yet efficient in the required number of processes.

Acknowledgments

We thank our anonymous referees for their helpful comments and advice. Flavio Junqueira has been partially supported by the INNCORPORA - Torres Quevedo Program from the Spanish Ministry of Science and Innovation, co-funded by the European Social Fund.

6. REFERENCES

[1] A. S. Aiyer, L. Alvisi, R. A. Bazzi, and A. Clement. Matrix Signatures: From MACs to Digital Signatures in Distributed Systems. In *22nd International Symposium on Distributed Computing (DISC '08)*, pages 16–31, 2008.

[2] G. Avoine and S. Vaudenay. Optimal Fair Exchange with Guardian Angels. In *Proc. 4th International Workshop Information Security Applications (WISA '03)*, pages 188–202, 2003.

[3] G. Bracha. An Asynchronous [(n-1)/3]-Resilient Consensus Protocol. In *Proc. 3rd ACM Symposium on Principles of Distributed Computing (PODC '84)*, pages 154–162, 1984.

[4] G. Bracha. Asynchronous Byzantine Agreement Protocols. *Information and Computation*, 75(2):130–143, 1987.

[5] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[6] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43:225–267, March 1996.

[7] B.-G. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz. Attested append-only memory: making adversaries stick to their word. In *21st ACM Symposium on Operating Systems Principles (SOSP '07)*, pages 189–204, 2007.

[8] B. A. Coan. A Compiler that Increases the Fault Tolerance of Asynchronous Protocols. *IEEE Trans. Computers*, 37(12):1541–1553, 1988.

[9] M. J. Fischer, N. A. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. In *Proc. 4th ACM Symposium on Principles of Distributed Computing (PODC '85)*, pages 59–70, 1985.

[10] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of ACM*, 32(2):374–382, 1985.

[11] M. Fort, F. C. Freiling, L. D. Penso, Z. Benenson, and D. Kesdogan. TrustedPals: Secure Multiparty Computation Implemented with Smart Cards. In *Proc. 11th European Symposium on Research in Computer Security (ESORICS '06)*, pages 34–48, 2006.

[12] V. Hadzilacos and S. Toueg. A Modular Approach to Fault-Tolerant Broadcasts and Related Problems. Technical report, Ithaca, NY, USA, 1994.

[13] C. Ho, D. Dolev, and R. van Renesse. Making Distributed Applications Robust. In *11th International Conference Principles of Distributed Systems (OPODIS '07)*, pages 232–246, 2007.

[14] C. Ho, R. van Renesse, M. Bickford, and D. Dolev. Nysiad: Practical Protocol Transformation to Tolerate Byzantine Failures. In *4th USENIX Symposium on Networked Systems Design and Implementation (NSDI '08)*, pages 175–188, 2008.

[15] L. Lamport, R. E. Shostak, and M. C. Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[16] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda. TrInc: Small Trusted Hardware for Large Distributed Systems. In *6th USENIX Symposium on Networked Systems Design and Implementation (NDSI '09)*, pages 1–14, 2009.

[17] M. C. Pease, R. E. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM*, 27(2):228–234, 1980.

[18] P. E. Veríssimo. Travelling through wormholes: a new look at distributed systems models. *SIGACT News*, 37(1):66–81, Mar. 2006.

APPENDIX

A. RELIABLE BROADCAST PROTOCOLS

We consider two reliable broadcast protocol in the asynchronous setting from the literature. In Figure 3, we present a r-broadcast protocol [12] in the crash fault model for $n \geq f+1$, while in Figure 4, we include a r-broadcast protocol [3] in the Byzantine model for $n \geq 3f+1$.

broadcast protocol for process p_i during session (τ, s)

upon a message $(\tau, \texttt{in}, \textsf{r-broadcast}, m)$ /* for the sender s*/
  for all $j \in [1, n]$ **do**
    send the message $(\tau, s, \texttt{send}, m)$ to p_j
  end for
  output $(\tau, s, \texttt{out}, m)$

upon a message $(\tau, s, \texttt{send}, m)$ from p_m for the first time:
  for all $j \in [1, n]$ **do**
    send the message $(\tau, s, \texttt{send}, m)$ to p_j
  end for
  output $(\tau, s, \texttt{out}, m)$

Figure 3: Reliable Broadcast for $n \geq f + 1$ for the crash Model [12]

broadcast protocol for process p_i during session (τ, s)

upon initialization:
  for all m **do**
    $e_m \leftarrow 0$; $r_m \leftarrow 0$
  end for

upon a message $(\tau, \texttt{in}, \textsf{r-broadcast}, m)$ /*for the sender s*/
  for all $j \in [1, n]$ **do**
    send the message $(\tau, s, \texttt{send}, m)$ to p_j
  end for

upon a message $(\tau, s, \texttt{send}, m)$ from s for the first time:
  for all $j \in [1, n]$ **do**
    send the message $(\tau, s, \textsf{echo}, m)$ to p_j
  end for

upon a message $(\tau, s, \textsf{echo}, m)$ from p_ℓ for the first time:
  $e_m \leftarrow e_m + 1$
  if $e_m = \lceil \frac{n+f+1}{2} \rceil \bigwedge r_m < f + 1$ **then**
    for all $j \in [1, n]$ **do**
      send the message $(\tau, s, \textsf{ready}, m)$ to p_j
    end for
  end if

upon a message $(\tau, s, \textsf{ready}, m)$ from p_ℓ for the first time:
  $r_m \leftarrow r_m + 1$
  if $r_m = f + 1 \bigwedge e_m < \lceil \frac{n+f+1}{2} \rceil$ **then**
    for all $j \in [1, n]$ **do**
      send the message $(s, \tau, \textsf{ready}, m)$ to p_j
    end for
  else if $r_m = n - f$ **then**
    output $(\tau, s, \texttt{out}, m)$
  end if

Figure 4: Bracha's Reliable Broadcast for $n \geq 3f + 1$ for the Byzantine Fault Model [3]

B. A CORRECTNESS ARGUMENT FOR THE GENERIC TRANSLATION

We present an argument that the translation satisfies the safety, integrity, and consistency properties. We note that many crash fault tolerant protocols also require liveness; i.e, if a correct sender p_i sends a message to a correct process p_j, then process p_j will receive the message eventually. As we discussed in Section 4.1, this requirement is satisfied by the asynchronous communication model itself, e.g., our communication model defined in Section 2 for the r-broadcast primitive.

Safety.

Our (transferable) authentication mechanism ensures that a faulty process cannot impersonate a correct process. Therefore, if the sender p_i is correct and a correct process p_j receives a message m from p_i, then the sender p_i must have executed send with m.

Integrity.

Integrity requires that if a correct process p_j receives a message m, then m is a valid state machine message. It is provable using the transferable authentication mechanism in a recursive manner as follows:

In our translated system, every sender p_s attaches a proof of correctness in the form of transferable authentication tokens to a message sent to a receiver p_r. The transferable authentication tokens include the corresponding messages ($\mathcal{R}_i$ for $i \in [1, h]$) p_s received. These messages also carry authentication tokens σ_i for $i \in [1, h]$, which process p_r uses to verify the authenticity of $\mathcal{R}_i$. When these verification are successful, p_r sequentially runs the received messages $\mathcal{R}_i$ for $i \in [1, h]$ through a state machine Σ_{p_s} for p_s starting from the initial state $\Sigma_{p_s,0}$. If the received message m corresponds to the state $\Sigma_{p_s,h}$, p_r is ensured that m is a valid message.

Consistency.

Consistency requires that if correct processes p_1 and p_2 receive valid messages $m_i \in \mathcal{S}_i$ and $m_j \in \mathcal{S}_j$ respectively from process p_g, then either (a) $B_{p_g,i}$ is a prefix of $B_{p_g,j}$, (b) $B_{p_g,j}$ is a prefix of $B_{p_g,i}$, or (c) $B_{p_g,i} = B_{p_g,j} \wedge \mathcal{S}_i = \mathcal{S}_j$.

We ensure this properties using the non-equivocation mechanism. As we discuss in the pseudocode, every message sent by a sender p_g is registered in its local non-equivocation mechanism. This enforces a total order on the messages sent sender p_g. Jointed with integrity, this itself can be sufficient to consistency. However, we further simplify the consistency enforcement by mandating p_g to transmit all the previous sequence numbered messages along with its current message. This ensures that a receiver does not miss any past of messages. Resultantly, when two receivers p_1 and p_2 hear from the same sender p_g their behaviors corresponding to the sender p_g will follow the exact same state transition path. However, as p_g may not send all messages, a behavior of process (say) p_1 can form a prefix of a behavior by process p_2, and vice versa.