Duck Attack on Accountable Distributed Systems

Amrit Kumar*
National University of Singapore
Singapore
amrit.kumar@polytechnique.org

Cédric Lauradoux
Inria
France
cedric.lauradoux@inria.fr

Pascal Lafourcade
Université Clermont Auvergne
France
pascal.lafourcade@uca.fr

ABSTRACT

Accountability plays a key role in dependable distributed systems. It allows to detect, isolate and churn malicious/selfish nodes that deviate from a prescribed protocol. To achieve these properties, several accountable systems use at their core cryptographic primitives that produce non-repudiable evidence of inconsistent or incorrect behavior. In this paper, we show how colluding adversaries can exploit the use of cryptographic digests in accountability protocols to mount what we call a *duck attack*. In a duck attack, colluding adversaries exploit the use of cryptographic digests to alter the transmission of messages while still looking honest: selfish behaviors remain undetected. We first discover the duck attack while analyzing PAG — a custom cryptographic protocol for accountable systems presented at ICDCS 2016. We later discover that accountable distributed systems based on tamper-evident log are also vulnerable to the duck attack and apply it on AcTinG — a protocol presented at SRDS 2014. To defeat our attack, we modify the commonly used hash-based log structure to have high-order dependency level between the authenticators and the messages stored in the log.

KEYWORDS

ACM proceedings, LaTeX, text tagging

ACM Reference format:
Amrit Kumar, Cédric Lauradoux, and Pascal Lafourcade. 2017. Duck Attack on Accountable Distributed Systems. In *Proceedings of International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, Melbourne, Australia, November 2017 (MobiQuitous'17),* 8 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Distributed systems are often plagued by nodes exhibiting selfish and malicious behavior. Selfish (rational) nodes may deviate from the prescribed protocol as and when there is an incentive to do so. Malicious or byzantine nodes can deviate arbitrarily from the protocol without having any well defined incentive. The end result is that the quality of the service to honest nodes gets affected and very often the honest nodes do not get served at all.

To this end, several approaches [HKD07, LCM+08, BDHU09, GHK+10, MDQ14a] have been proposed in the past to force these "rebellious" nodes to be compliant (to the underlying protocol) and to make them *accountable* for their actions in the network. By making every node accountable, nodes do not have incentive to deviate as any such attempt will eventually be detected resulting in its churn out from the network.

While considering accountability, attacks resulting from collusion between nodes are certainly hard to thwart as evidenced in [ELW11, WA13]. This is because colluding nodes generally perform unobservable actions from the point of view of the protocol making their deviations difficult to deter. In fact, this is also why the number of accountable protocols capable of handling colluding nodes is rather limited. FlightPath [LCM+08] fights collusion using Tit-for-Tat incentives. LiFTinG [GHK+10] uses cross checking and statistical analysis. Finally, AcTinG [MDQ14a] and PAG [DMPQ16] rely on cryptography to ensure that nodes' actions are bound, non-repudiable, tamper-evident and verifiable [YC04, YC05]. This last category of solutions is the focus of our work. We analyze the security provided by AcTinG [MDQ14a] and PAG [DMPQ16] and discover a new attack.

Contributions: The main contribution of this paper is the design of a new form of collusion attack: the *duck attack*. The duck attack is named after the duck test[1] and it can be summarized as follows: *if you provide me the cryptographic digest $H(m)$ of a message m, then it probably means that you know $H(m)$ and the message m.* Note that, this is outrightly false because **concluding that you know m from the fact that you know $H(m)$ is incorrect**. We find this logical flaw in PAG [DMPQ16] and AcTinG [MDQ14a]. In a duck attack, the colluding nodes execute the protocol without exchanging the messages. Instead, they exchange the message digests and later commit the messages, when it is more advantageous to do so. For instance, they can use data compression techniques to send batches of messages and therefore save bandwidth. The duck attack is very similar to terrorist fraud [ABK+11] and distance highjacking attacks [CRC12] against authentication and distance bounding protocols, where colluding attackers attempt to confuse an authentication server.

To fight the duck attack, we revisit the *tamper-evident log structure* of AcTinG, which is a secure log initially proposed in [Pet02] and later also employed in several other protocols such as in PeerReview [HKD07]. Our approach consists in modifying the secure log. In AcTinG, a secure log is maintained by each node and each entry (message sent by the node) of the secure log yields an *authenticator*. The authenticator attests the creation of the log entry and can be publicly verified to detect deviation from the protocol. In our countermeasure, the computation of the authenticator *depends on all the previous messages* stored in the secure log. Cheating with the duck attack is prevented because one of the colluding parties

*Work done while at Université Grenoble Alpes and Inria, France

MobiQuitous'17, Melbourne, Australia

DOI: 10.1145/nnnnnnn.nnnnnnn

[1] https://en.wikipedia.org/wiki/Duck_test

cannot maintain the log only by herself: She needs to make a lot of communication with her accomplice.

Outline: The next section of this paper is dedicated to the presentation of the three protocols that we attack in this paper. In Section 2, we discuss the threat model and colluding adversaries. We first demonstrate the power of the duck attack on PAG [DMPQ16] (Section 3). In Section 4, we also illustrate the duck attack on AcTinG a protocol which inherits the secure log system of PeerReview [HKD07]. Our log system with extended dependency is presented in Section 5 to thwart the attack. Finally in Section 6, we conclude the paper and mention some future works.

2 THREAT MODEL

We consider a network composed of correct nodes, individual (non colluding) malicious nodes and malicious colluding nodes. Correct nodes respect the normal execution of the protocol. Individual malicious nodes tamper with the execution of the protocol. They can modify the messages they receive and forward the modified version to the rest of the network for instance. Malicious colluding nodes can behave exactly as individual malicious nodes to harm the system. The colluding nodes can also have a rational behavior (see [AAC+05]). They follow the protocol if it is not in their interest but they are willing to deviate from it if they can save resources like bandwidth or computation. In order to do so, they can communicate using off-the-record channels or use covert channels [ELW11] to exchange data beyond those prescribed by the underlying protocol.

We assume throughout the paper that nodes are set up with some long term asymmetric cryptographic keys. We assume that nodes do not share their long term private keys with anybody. This is a limit to which the colluding nodes can share information to cheat. This assumption is implicitly made in most papers like PeerReview [HKD07]. Without it, colluding nodes sharing secrets would be equivalent to a single adversary controlling two nodes. A similar assumption is made in the analysis of authentication protocol resistant to certain form of collusion [ABK+11, CRC12].

3 DUCK ATTACK ON PAG (AND MORE)

PAG [DMPQ16] has been designed to provide accountability to fight both individual malicious nodes and malicious colluding nodes in *gossip-based message dissemination schemes*, where nodes periodically exchange data chunks with randomly chosen nodes [KMG03, BMM+08]. In addition, it provides privacy guarantees for the messages provided by the nodes. As our duck attack does not take advantage of the privacy mechanism, privacy properties are not detailed here and left in Appendix. The appendix also provides attacks on the privacy guarantees of PAG.

The main goal of PAG is to ensure that nodes respect the *obligation-to-receive* and the *obligation-to-forward* which are defined as (see [DMPQ16]):

- **Obligation-to-receive**: At a given communication round, a node must receive the messages sent by its predecessors that it received.
- **Obligation-to-forward**: A node must forward the messages it received at a given communication round R to all its successors during round $R + 1$.

We describe how PAG enforces these obligations.

3.1 PAG's Description

3.1.1 Assumption and setup. Consider a simple network composed of five nodes: A, B, C, D and E (see Fig. 1). The point of interest in this network is D which receives messages (in the form of updates) from the producers A, B and C. In practice, A, B and C may just forward the messages received from a previous hop rather than creating them. The expected task of D is to forward the messages to E. The nodes A, B and C are called *predecessors* of D, while the node E is referred to as a *successor* of D.

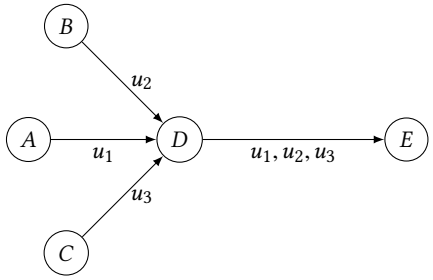

Figure 1: Gossip based message dissemination.

PAG assumes secure channels between the nodes: all node-to-node communications are encrypted with the help of a public-key infrastructure. Messages are also digitally signed to ensure their authenticity. However, in order to ease the presentation, we drop all node-to-node encryptions and the accompanied signature.

The core primitive used in PAG is a *keyed homomorphic hash function*. The function allows a *witness* (an auditor) to perform the privacy preserving verification. Following the notation used in [DMPQ16], we denote this hash function by H_p, where p is the key. The hash function is homomorphic in the sense that for any two messages u and v, and keys p, p_1 and p_2, the following two properties hold:

$$\begin{aligned} H_p(u) \cdot H_p(v) &= H_p(u \cdot v), \\ H_{p_1}(H_{p_2}(u)) &= H_{p_1 \cdot p_2}(u). \end{aligned}$$

In PAG, H_p is instantiated by a variant of the RSA function with a prime exponent p and a modulus M. Hence, for any message $u \in \{0,1\}^*$, the digest of the hash function H_p is given as:

$$H_p(u) = u^p \bmod M.$$

The modulus is chosen once at the start of the protocol, while a different key (the exponent) is used for every predecessor node. The keys are also updated at the start of each round.

3.1.2 Protocol Core. We continue with the example network of Fig. 1 with six nodes: A, B, C, D, and E, where D is the point of interest. It has three predecessors A, B and C and a successor E. We augment this network with a witness W for D.

The protocol runs in rounds and assumes that the network is synchronous. A schematic representation of the messages exchanged in a round of PAG is given in Fig. 2. Step 1, each predecessor node asks for a key. D hence generates three random keys p_1, p_2, p_3 and sends $p_1, p_2 \cdot p_3$ to A; $p_2, p_1 \cdot p_3$ to B and $p_3, p_1 \cdot p_2$ to C. Step 2,

the predecessor nodes send their respective messages u_1, u_2 and u_3 to D. Step 3, each predecessor node computes the homomorphic hash of its message with the key received from D and sends to the witness the computed digest along with the received product of two keys. For instance, A sends $H_{p_1}(u_1), p_2 \cdot p_3$ to W. Meanwhile, upon reception of the messages from its predecessors, D forwards them to its successor E along with the product of all the keys. E then computes $H_{p_1 \cdot p_2 \cdot p_3}(u_1 \cdot u_2 \cdot u_3)$ and sends the digest to W.

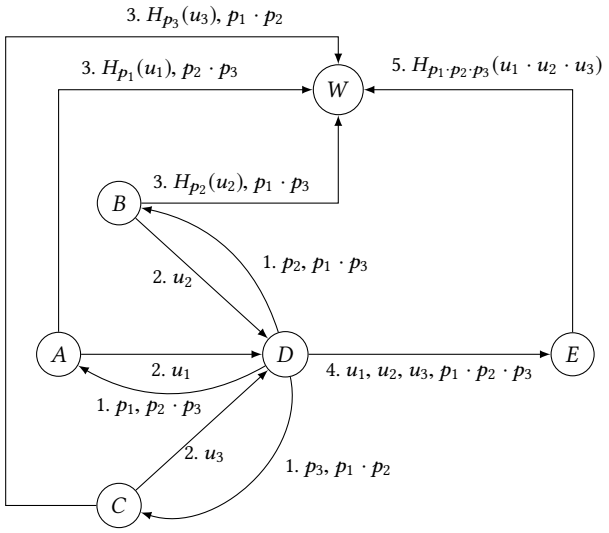

Figure 2: A round of PAG with three predecessors (A, B, C) and one successor (E). W is a witness for D.

Upon reception of all the messages from the predecessors and the successor, W computes: $x = H_{p_2 \cdot p_3}(H_{p_1}(u_1))$, $y = H_{p_1 \cdot p_3}(H_{p_2}(u_2))$, and $z = H_{p_1 \cdot p_2}(H_{p_3}(u_3))$. Values x, y and z are computed using the messages received from A, B and C respectively. The witness then checks for equality between $x \cdot y \cdot z$ and $H_{p_1 \cdot p_2 \cdot p_3}(u_1 \cdot u_2 \cdot u_3)$ (the last digest was sent by E). If the messages were correctly forwarded, the homomorphic properties of the hash function ensure that the two terms are equal. Indeed,

$$\begin{aligned} x \cdot y \cdot z &= H_{p_2 \cdot p_3}(H_{p_1}(u_1)) \cdot H_{p_1 \cdot p_3}(H_{p_2}(u_2)) \cdot H_{p_1 \cdot p_2}(H_{p_3}(u_3)) \\ &= H_{p_1 \cdot p_2 \cdot p_3}(u_1) \cdot H_{p_1 \cdot p_2 \cdot p_3}(u_2) \cdot H_{p_1 \cdot p_2 \cdot p_3}(u_3) \\ &= H_{p_1 \cdot p_2 \cdot p_3}(u_1 \cdot u_2 \cdot u_3) \end{aligned}$$

If the equality test fails, then W may conclude that D did not correctly forward the messages to E.

We note that the actual protocol is slightly more complicated since it assumes several witnesses for a given node and that the node E does not directly communicate with the witness. In fact, the witnesses for D and E communicate with each other to crosscheck the messages sent by the monitored nodes. The protocol simplification is only meant to ease the understanding of the presented attacks. It does not limit in any manner the scope of the attacks on the actual protocol.

3.2 Key Recovery Attack

It is argued in [DMPQ16] that in order to ensure the privacy *preserving auditing* of the nodes, the witness should not learn the keys generated by a node. Moreover, it is claimed in [DMPQ16] that if every node has three predecessors, then the witness can not learn the keys. We first show that this claim is unfounded by illustrating an attack on our example network of three predecessors Fig. 2. The goal of this attack is to further simplify the PAG protocol so that the presentation of our duck attack becomes easier.

The witness W receives $p_1 \cdot p_2$ from C, and $p_1 \cdot p_3$ from B (*Cf.* Fig. 2). In order to recover the key p_1, it simply computes $\gcd(p_1 \cdot p_2, p_1 \cdot p_3)$. Since, the keys are prime numbers and generated randomly, this computation yields p_1. Other keys p_2 and p_3 can be obtained likewise. It is evident that this attack trivially extends to any network with arbitrary number of predecessors.

As a result, all the keys can eventually be recovered by the witness. Hence, the keyed hash function now reduces to an unkeyed hash function. Apart from being an attack, this result allows us to further simplify the protocol by assuming that instead of generating one key per predecessor, nodes create a unique key p per round. The key is public to any node participating in the network (valid under the assumption that nodes collude). This does not change the verification procedure at the witness node. In the rest of this section, we work with only two predecessors to simplify the presentation.

3.3 Duck Attack with Colluding Nodes

We now present the duck against PAG, where, a unique key known to the witness node is used. To this end, we assume the message transmission scenario as depicted in Fig. 3, where two predecessor nodes A and B send messages u and v respectively to C. The node W is the witness for C. We assume that nodes C and D collude. Hence, to save bandwidth C sends to D the digest $H_p(v)$ instead of the message v. Since, D colludes with C, it does not raise alarm and sends $H_p(u \cdot v)$ to the witness W. The witness can not detect the discrepancy and declares the verification as successful. Hence, PAG does not ensure the obligation-to-forward guarantee.

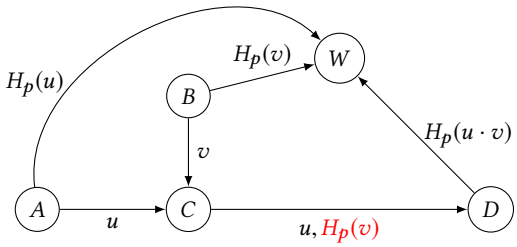

Figure 3: Duck attack on a round of PAG. C instead of forwarding the message v to D, forwards $H_p(v)$. In red, we show the message sent as an adversary.

Another variant of the duck attack is possible. To this end, we consider a situation where a predecessor node (B) and the receptor node (C) are malicious and collude to save bandwidth. Under this setting, we consider the message transmission scenario as depicted in Fig. 4, where two predecessor nodes A and B wish to send the

same message u to C. The latter may inform B before the transmission of the actual message that it is about to receive or (has received) the same message from another of its predecessors. Hence, to save bandwidth B does not send the message to C. However, it sends $H_p(u)$ to W to convince the witness that it in fact went through with the transmission. C colludes with B and forwards two copies of u to D and the rest of the protocol round runs as any other. The protocol does not respect the obligation-to-forward.

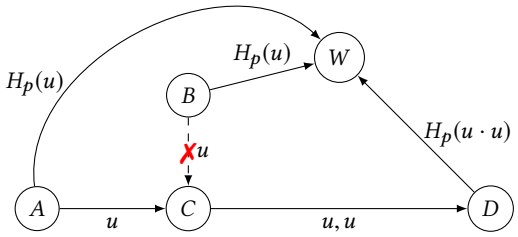

Figure 4: Collusion attack on a round of PAG. Dashed arrow and the symbol ✗ is to denote that this transmission never occurred.

Clearly, with the existing mechanism in PAG, it is not possible for witnesses to detect such behaviors. The fundamental reason why this selfish behavior goes undetected in PAG is that the witness has no way to learn that the digest sent by D on the message u did originate from B.

4 DUCK ATTACK ON ACTING

AcTinG [MDQ14a] is a protocol presented at SRDS 2014 and inspired by PeerReview [HKD07] and FullReview [DMAQ14]. These protocols rely on maintaining a secure data structure on each node which implements a tamper evident log. The log is used to record the messages a node has sent and received. Eventually, any node (verifier) may request the log of another node (prover) and independently determine whether the prover has deviated from its expected behavior.

AcTinG has several layers, we first describe how two nodes interact when a message is exchanged and how the duck attack applies to this exchange. We then explain how the colluding adversaries can predict when the audits take place in AcTinG.

4.1 Secure Log in AcTinG

Each node in AcTing maintains a secure log. Let us assume that at time i, node A has last entry $T_i||i||m_i$ which contains an *authenticator* (later used for audit) T_i, a sequence number i (a monotonic counter) and a message m_i sent by it. Then, the next entry $T_{i+1}||i+1||m_{i+1}$ is defined by:

$$T_{i+1} = H(T_i||i+1||H(m_{i+1})),$$

with H a cryptographic hash function and T_0 a fixed value. Note that the authenticator chains all previous messages sent and generates a single digest. Checking the validity of an authenticator means checking that all previous messages have been correctly logged and that the log has not been tampered with.

We have simplified how AcTinG stores messages in the log. In fact, AcTing uses a special format and structure for the message which is not relevant to our attack. Fig. 5 describes how the log entries are created during a communication between two nodes A and B. In addition to the message m, node A sends:

$$\alpha^A_{i+1} = \sigma(i+1||T^A_{i+1}),$$

with σ a digital signature algorithm. The value α^A_{i+1} is used by B during the audit to check the correctness of A.

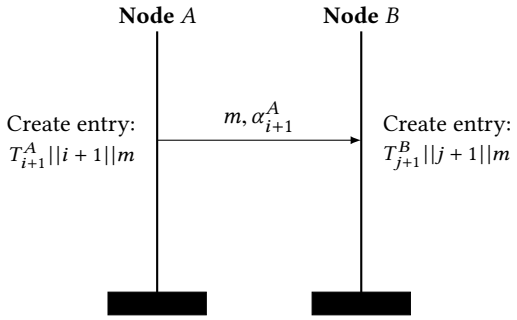

Figure 5: Log entries created by AcTinG during the transmission of m by A to B.

4.2 Luring AcTinG

The duck attack against AcTinG is based on the observation that node B does not need to know m to compute T^B_{j+1} but only $H(m)$. Node A just needs to send $H(m)$ to B. Then, B can create a blank entry $T^B_{j+1}||j+1||\emptyset$ which does not contain the message but has a valid authenticator T^B_{j+1} that can be used to compute T^B_{j+2}. Node B can continue to create new entries in its log despite the absence of m. If a node performs an audit of B, then it will discover the blank entries. Therefore, it is important that before an audit, A sends m to B. The attack is schematically presented in Fig. 6.

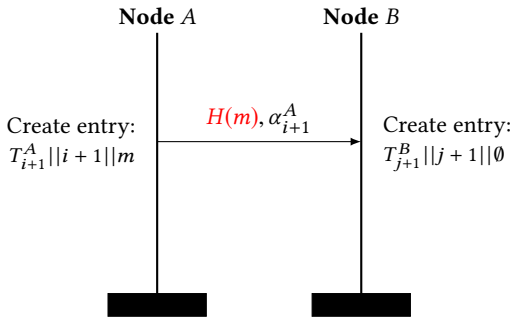

Figure 6: Duck attack for A and B.

At the first sight, the duck attack may appear to be pointless: executing the attack strategy is more costly than following the protocol. In the duck attack, node A sends $H(m)$ and then must late

commit m to B. The communication of $H(m)$ is an extra overhead compared to the normal execution of the protocol. The duck attack is not interesting if the colluding nodes execute it for a single message. The attack is interesting if they use it for several messages. Let us assume that node B has created two blank entries with the help of A. Each entry is associated with message m and m'. Node A needs to commit m and m' before the audit of B. Node A can use data compression to send m and m' with gain bounded by information theory argument and reduce the cost of communicating with B. With many blank entries, the colluding nodes can even expect greater gain.

4.3 Predicting AcTinG's Audit

It is now crucial for the nodes who want to mount a duck attack to know when the audit will be performed. When a node A is associated to a node B, node A decides if it performs a full audit of B or not. The full audit of B implies to check:

- the secure log of node B;
- the secure logs of all the nodes which have communicated with B;
- B has forwarded all the messages it was supposed to;
- B has performed all the necessary audits.

The decision to run this audit is taken using a *non deterministic process*:

- Node A chooses a signature α_i and computes:

$$r = \alpha_i \bmod 100$$

- Node A computes:

$$r = H(PK_A||PK_B||i) \bmod 100,$$

with PK_A and PK_B the respective public key of A and B and i the current value of the counter.

If the value r is greater than a certain threshold then A audits B. The paper suggests a threshold of 30%.

The first solution favors selfish nodes. A chooses α_i such that no audit takes place. The second solution is deterministic and predictable. Nodes A and B can compute in advance all the values $H(PK_A||PK_B||i)$ for all the nodes to determine when they are going to be audited. Therefore, they can determine when they can execute the duck attack without any risk of being caught. The conjunction of the duck attack and the fact that the audit of AcTinG is predictable makes the risk analysis presented AcTinG [MDQ14a] too optimistic about the probability to detect a fraud.

5 COUNTERMEASURES

The authors of this paper do not believe it is possible to fix PAG to achieve both security and privacy. Additional security and privacy issues of PAG are presented in Appendix to support our opinion. In this section, we focus on fixing AcTinG. We propose two modifications of AcTinG to defeat the duck attacks: the usage of verifiable pseudo-random function and a modified secure log data structure.

In AcTinG, the nodes need to use randomness to decide when the audit are made and to choose the nodes they communicate with. In the current description of AcTinG, the random number generator is either deterministic or not accountable. It emphasizes the difficulty to achieve accountability when the nodes have a non-deterministic behavior. This problem was first tackled by Backes *et al.* [BDHU09] to extend PeerReview to non-deterministic nodes' behavior. The random number used during the protocol must be unpredictable and verifiable. This property was achieved in CSAR [BDHU09] by using verifiable pseudo-random functions [MRV99]. It is the natural option to fix AcTinG. Guerraoui *et al.* [GHK+10] have proposed a different approach by measuring entropy, but the guarantees are not as strong as those provided by CSAR. Using a verifiable pseudo-random function ensures that adversaries can not predict or influence the result of the random number generator without being detected.

AcTinG [MDQ14b], PeerReview [HKD07] and many others [BDHU09, DMAQ14] are based on the secure log structure proposed by Maniatis and Baker in [Pet02]. The security of their design is based on one-way hash function H (SHA-1 en 2002). Maniatis and Baker in fact propose a *secure timeline log* in [Pet02]. At time i, the last entry in the log contains message m_i and is associated with authenticator T_i. Then, the next entry is associated to T_{i+1} defined by:

$$T_{i+1} = H(i+1||T_i||G(m_{i+1})).$$

T_0 is a fixed value. The function G is also a one-way hash function. The security argument given to justify their design works as follows. Given $T_{i+1} = H(i+1||T_i||G(m_{i+1}))$, it is not possible to produce a message β and an authenticator $T'_i \neq T_i$ such that $T_{i+1} = H(i+1||T'_i||G(\beta))$ since H is second pre-image resistant.

The authors of [Pet02] justified the use of G in their construction by the fact that the m_i can be very large (complete Merkle's tree or any authenticated data structure). It is possible without loss of security to get rid of the function G as we assume now that H is a collision-resistant hash function (SHA-3 [Dwo15]). In our scheme, we compute the authenticator as follows:

$$T_{i+1} = H(i+1||T_i||m_{i+1}||m_i||\cdots||m_1)).$$

The computation of T_{i+1} depends on all the previous messages stored in the log. . For a given $T_{i+1} = H(i+1||T_i||m_i||m_{i-1}||\cdots||m_1))$, it is intractable for an adversary to find $T'_i \neq T_i$ and/or $\alpha \neq m_i$ such that:

$$T_{i+1} = H(i+1||T'_i||m_{i+1}||\alpha||\cdots||m_1)),$$

because H is collision resistant. Therefore our modification is secure.

Let us consider two nodes A and B which exchange messages and maintain a log of their communication using our modification. We also assume that each time a node B receives a message m_i from A, it sends to A an acknowledgement $T_i^B||\sigma(T_i^B)$ with $\sigma(T_i^B)$ a cryptographic signature. It appears that the duck attack still applies. The only difference with the duck attack previously described in this paper is that node B needs to provide to A the current authenticator of its log (see Fig. 7). At the end, B sends its signature to A to end the communication. The use of a randomized signature does not make the attack more difficult because A can recompute everything for B (except the final signature).

Our approach is not to detect or prevent directly the duck attack but to make it more costly than any benefit that two colluding parties can expect. Let consider a case in which node B colludes with node A and exchanges messages with an honest node C. This situation is represented in Fig. 8. A and B first execute the duck

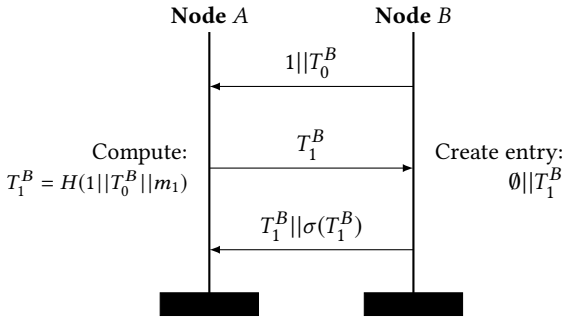

Figure 7: Duck attack on the modified secure log structure.

attack to avoid sending message m_1. Node A computes and sends T_1^B to B. Then, node B receives message m_2 from C. Without the knowledge of m_1, node B cannot compute T_2^B and cannot send the acknowledgement to C. To solve this problem, there are two options for the colluding nodes: A sends m_1 to B or B sends m_2 to A and wait to receive T_2^B. If A sends m_1 to B, they have exchanged more data than if they had followed the protocol. The same conclusion holds if B sends m_2 to A. This problem occurs each time B must log message from an honest node.

Node B can be tempted to acknowledge the reception of m_2 with a random value $r||\sigma(r)$. However, an audit of B and C will detect this action. Note that maintaining multiple logs per partner is an attack out-of-the-scope of this paper. This issue has been tackled in [HKD07] for instance.

Fig. 8 clearly shows the benefit of increasing the dependency order of the authenticators. If node B tries to create a blank entry with A, it affects the computation of the next authenticators obtained from an honest node. An issue of our modification is that the complexity to compute the authenticators in the log grows linearly with the log size. It is possible to find a trade-off between complexity and security by using a window of Δ previous entries:

$$T_{i+1} = \begin{cases} H(i+1||T_i||m_{i+1}||m_i||\cdots||m_{i-\Delta+1}) & \text{if } i \geq \Delta \\ H(i+1||T_i||m_{i+1}||m_i||\cdots||m_1) & \text{if } 0 < i < \Delta \\ H(i+1||T_i||m_{i+1}) & \text{if } i = 0 \end{cases}$$

We have computed the time to create a secure log of 100 entries in `Python 2.7.12` on an Intel Core i7-5500U (2.40GHz) processor . The size of each message is 10KB (arbitrarily chosen). We have used SHA-3 using 512-bit digests. When the computation of authenticators depends on all the previous entries, we observe a slowdown of $\times 64$ compared to the original secure log presented in [Pet02]. Fig. 9 shows that the slowdown grows linearly with the value of Δ. The value $\Delta = 1$ has only a small impact on the performance and can prevent the duck attack if there is a high probability that two consecutive messages are never sent by the same source. This can be achieve by randomly choosing the source in each round.

6 CONCLUSION

We have applied the duck attack on a custom cryptographic system PAG and on AcTinG based on secure log. The origin of the attack is the use of cryptographic hash function as a core of more complex primitives. It allows attackers to exchange digests instead of the real messages and therefore be selfish. After discovering our attacks on PAG, we have been unable to propose corrections that fix all the flaws. It seems difficult to do so for PAG. We believe that designing a protocol that guarantees the accountability property and at the same time respects the privacy of the nodes might be very challenging. Designs based on a secure log can however be fixed by increasing the dependency order of the authenticators' computation and by using verifiable pseudo-random function.

A key issue to go forward in the design of accountable distributed protocols is the definition of the selfish adversaries. Recently in [KTV10] *Kuesters et al.* presented a formal definition of accountability and some possible links with verifiability properties. It might be interesting to see how these definitions are sensitive to the duck attack and how they match selfish adversaries. In other words, we can investigate whether the two protocols studied in this paper satisfy their definitions or if the duck attack allows us to prove the contrary.

ACKNOWLEDGMENT

The work was partly supported by the LabEx PERSYVAL-Lab (ANR–11-LABX-0025), the project-team SCCyPhy and the Digital Trust Chair from the University of Auvergne Foundation.

REFERENCES

[AAC+05] Amitanand S. Aiyer, Lorenzo Alvisi, Allen Clement, Michael Dahlin, Jean-Philippe Martin, and Carl Porth. BAR Fault Tolerance for Cooperative Services. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles 2005, SOSP 2005, Brighton, UK, October 23-26, 2005*, pages 45–58, New York, NY, USA, 2005. ACM.

[ABK+11] Gildas Avoine, Muhammed Ali Bingöl, Süleyman Kardas, Cédric Lauradoux, and Benjamin Martin. A framework for analyzing RFID distance bounding protocols. *Journal of Computer Security*, 19(2):289–317, 2011.

[BDHU09] Michael Backes, Peter Druschel, Andreas Haeberlen, and Dominique Unruh. CSAR: A Practical and Provable Technique to Make Randomized Systems Accountable. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA, 8th February - 11th February 2009*, page 13, Virginia, USA, 2009. The Internet Society.

[BMM+08] Thomas Bonald, Laurent Massoulié, Fabien Mathieu, Diego Perino, and Andrew Twigg. Epidemic Live Streaming: Optimal Performance Trade-Offs. In *Proceedings of the 2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 2008, Annapolis, MD, USA, June 2-6, 2008*, pages 325–336, New York, NY, USA, 2008. ACM.

[BR15] Elaine B. Barker and Allen L. Roginsky. Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths. Technical report, National Institute of Standards and Technology (NIST), nov 2015.

[CDL+00] S. H. Cavallar, B. Dodson, A. K. Lenstra, W. M. Lioen, P. L. Montgomery, B. Murphy, H. J. J. te Riele, K. Aardal, J. Gilchrist, G. Guillern, P. C. Leyland, J. Marchand, F. Morain, A. Muffet, C. Putnam, C. Putnam, and P. Zimmermann. Factorization Of A 512-Bit RSA Modulus. In B. Preneel, editor, *Advances in Cryptology*, volume 1807 of *Lecture Notes in Computer Science*, pages 1 – 18, California, USA, 2000. Springer.

[CRC12] Cas J. F. Cremers, Kasper Bonne Rasmussen, and Srdjan Capkun. Distance hijacking attacks on distance bounding protocols. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, pages 113–127, California, USA, 2012. IEEE.

[DMAQ14] Amadou Diarra, Sonia Ben Mokhtar, Pierre-Louis Aublin, and Vivien Quéma. FullReview: Practical Accountability in Presence of Selfish Nodes. In *33rd IEEE International Symposium on Reliable Distributed Systems, SRDS 2014*, pages 271–280, Nara, Japan, October 2014. IEEE Computer Society.

[DMPQ16] Jeremie Decouchant, Sonia Ben Mokhtar, Albin Petit, and Vivien Quéma. PAG: Private and Accountable Gossip. In *36th IEEE International Conference on Distributed Computing Systems, ICDCS 2016, Nara, Japan, June 27-30, 2016*, pages 35–44, Nara, Japan, 2016. IEEE.

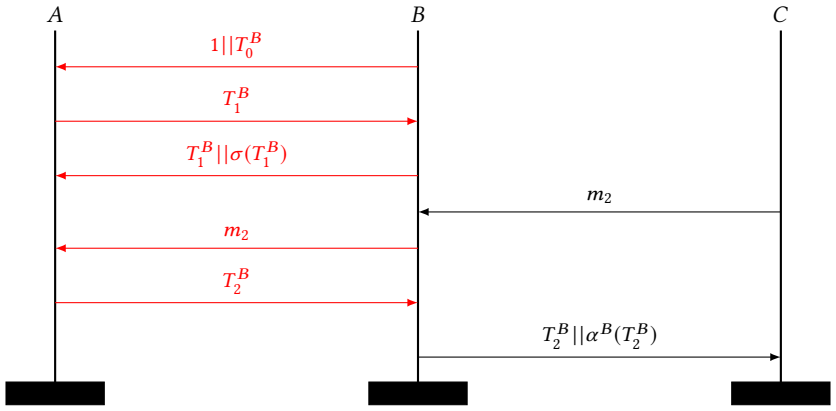

Figure 8: Duck attack with more participants.

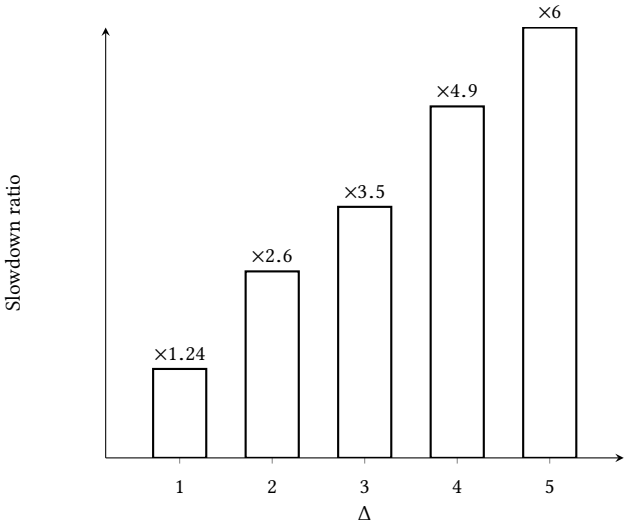

Figure 9: Slowdown observed for the creation of the log when the authenticators depends on the Δ previous entries.

[Dwo15] Morris J. Dworkin. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Technical Report FIPS 202, NIST, August 2015.

[ELW11] Raphael Eidenbenz, Thomas Locher, and Roger Wattenhofer. Hidden communication in P2P networks steganographic handshake and broadcast. In *30th IEEE International Conference on Computer Communications, INFOCOM 2011*, pages 954–962, Shanghai, China, April 2011. IEEE.

[GHK+10] Rachid Guerraoui, Kévin Huguenin, Anne-Marie Kermarrec, Maxime Monod, and Swagatika Prusty. LiFTinG: Lightweight Freerider-Tracking in Gossip. In *Middleware 2010 - ACM/IFIP/USENIX 11th International Middleware Conference*, volume 6452 of *Lecture Notes in Computer Science*, pages 313–333, Bangalore, India, November 2010. Springer.

[HKD07] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. PeerReview: Practical Accountability for Distributed Systems. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, pages 175–188, Washington, USA, 2007. ACM.

[KAF+10] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman J. J. te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-Bit RSA Modulus. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, pages 333–350, California, USA, 2010. Springer.

[KMG03] Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. Probabilistic Reliable Dissemination in Large-Scale Systems. *IEEE Trans. Parallel Distrib. Syst.*, 14(3):248–258, 2003.

[KTV10] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: definition and relationship to verifiability. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 526–535, Illinois, USA, 2010. ACM.

[LCM+08] Harry C. Li, Allen Clement, Mirco Marchetti, Manos Kapritsos, Luke Robison, Lorenzo Alvisi, and Michael Dahlin. FlightPath: Obedience vs. Choice in Cooperative Services. In *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008*, pages 355–368, San Diego, California, USA, December 2008. USENIX Association.

[MDQ14a] Sonia Ben Mokhtar, Jeremie Decouchant, and Vivien Quéma. AcTinG: Accurate Freerider Tracking in Gossip. In *33rd IEEE International Symposium on Reliable Distributed Systems, SRDS 2014*, pages 291–300, Nara, Japan, October 2014. IEEE Computer Society.

[MDQ14b] Sonia Ben Mokhtar, Jeremie Decouchant, and Vivien Quéma. AcTinG: Accurate Freerider Tracking in Gossip. In *33rd IEEE International Symposium on Reliable Distributed Systems, SRDS 2014, Nara, Japan, October 6-9, 2014*, pages 291–300, Nara, Japan, 2014. IEEE.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable Random Functions. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99*, pages 120–130, New York City, USA, 1999. IEEE Computer Society.

[Pet02] Petros Maniatis and Mary Baker. Secure History Preservation Through Timeline Entanglement. In *Proceedings of the 11th USENIX Security Symposium*, pages 297–312, San Francisco, USA, 2002. USENIX.

[WA13] Edmund L. Wong and Lorenzo Alvisi. What's a little collusion between friends? In *ACM Symposium on Principles of Distributed Computing, PODC '13*, pages 240–249, Montreal, QC, Canada, July 2013. ACM.

[YC04] Aydan R. Yumerefendi and Jeffrey S. Chase. Trust but Verify: Accountability for Network Services. In *Proceedings of the 11st ACM SIGOPS European Workshop, Leuven, Belgium, September 19-22, 2004*, page 37, Leuven, Belgium, 2004. ACM.

[YC05] Aydan R. Yumerefendi and Jeffrey S. Chase. The role of accountability in dependable distributed systems. In *Proceedings of the First Conference on Hot Topics in System Dependability*, HotDep'05, page 6, Berkeley, CA, USA, 2005. USENIX Association.

A MORE ATTACKS ON PAG

During our analysis of PAG, we discover several other weakness in PAG. Some of these weaknesses can be fixed but at the cost of some important properties for PAG (like privacy for instance). We discover that the homomorphic hash function used by the authors is not collision free.

A.1 Hash Collisions

PAG is also insecure because the cryptographic primitive used to instantiate the function H_p of the previous section has collisions. A trivial collision exists for two messages u and v such that $v = u + k \times M$ for any $k \in \mathbb{N}^*$:

$$\begin{aligned} H_p(v) &= v^p \bmod M, \\ &= (u + k \times M)^p \bmod M, \\ &= u^p \bmod M. \\ &= H_p(u). \end{aligned}$$

We note that the RSA function defined using a modulus M is a permutation on the set $[0, M-1]$. Hence, a hash function defined using it should not yield collisions when the message space is restricted to $[0, M-1]$. However, PAG assumes that any message u exchanged between the nodes is larger than M. This assumption clearly leads to collisions.

A.2 From Collisions to Downgrade Attack

We consider the network depicted in Fig. 10. For this network, we also consider the following situation during a round: Node A sends $u' = u + M$ to C and node B sends an arbitrary message v'. Node C can substitute u' by u without being detected. Indeed, C forwards u and v' to D. Then D sends $H_p(u \cdot v')$ to W. We recall our assumption that a unique p used throughout and is known to all the participants of the network.

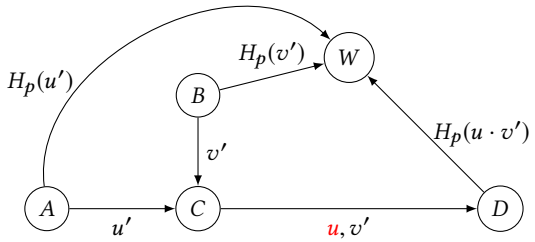

Figure 10: Downgrade attack on a round of PAG. In red, we show the message sent as an adversary.

The witness W upon receiving $H_p(u')$ and $H_p(v')$ from A and B respectively verifies that:

$$\begin{aligned} H_p(u') \cdot H_p(v') &= H_p(u \cdot v') \\ &= H_p(u) \cdot H(v'). \end{aligned}$$

W therefore concludes that C has indeed respected the *obligation-to-receive* and the *obligation-to-forward*. In reality though, C has not respected the *obligation-to-forward* by substituting u' by u. The node C has successfully downgraded u' to u by exploiting the collision.

A.3 Breaking Unlinkability

In this section, we present two attacks to break the unlinkability guarantee of PAG. Our first attack that we refer to as *short-term linkability* exploits the fact that PAG employs a small RSA modulus. Our second attack that we refer to as *long-term linkability* exploits our earlier observation that keys of the hash function can be learned by the witness, hence making the function behave deterministically.

A.3.1 Short-term Linkability. Contrary to the claim that PAG is preserving privacy, the witness can in fact invert the hash function and learn the message corresponding to a received digest. This allows the witness to link the message to the node that sent the digest and hence break the claimed unlinkability property.

Inversion of digests is possible since the hash function H_p is instantiated with a modulus M of size 512 bits. It is argued in [DMPQ16] that a 512-bit modulus suffices when the RSA function is used as a hash function. However, for a small modulus M of 512 bits, it is possible to obtain the prime factorization of M and compute $\varphi(M)$ where, φ is the Euler's totient function. Knowledge of $\varphi(M)$ allows to invert any digest $H_p(u)$ and obtain u as:

$$H_p(u)^{p^{-1} \bmod \varphi(M)} \bmod M = u.$$

It is to note that an RSA modulus of 512 bits named RSA-155 was successfully factored[2] by Herman te Riele *et al.* [CDL+00] as early as in 2000. The most recent RSA modulus factored is of 768 bits. The feat was achieved by Kleinjung *et al.* in 2010 [KAF+10]. In 2015, NIST published a standard [BR15] which recommends using a modulus of 2048 bits. These references clearly suggest that the choice of modulus size in PAG is inappropriate.

The linkability attack based on inverting the hash function holds as long as the modulus size is small enough. Choosing an appropriately large modulus prevents inversion. We hence refer to this attack as a *short-term linkability* attack.

A.3.2 Long-term Linkability. Choosing a sufficiently large modulus however does not prevent other linkabiliy attacks that the witness may mount. In fact, since the key of the hash function is known, the function now becomes deterministic. As a result, the witness can detect if a message is replayed by a node. It is also possible to relate two nodes if they send the same message. We refer to these attacks as *long-term linkability* attacks since they exist even when the modulus size is sufficiently large.

In order to understand the ensuing implications, let us consider a content sharing PAG network. If two different nodes send the same digest to a witness, then it may learn the *interest graph* between nodes sharing similar interests, thus possibly inferring private information about them.

[2] RSA-155 was a part of the RSA Factoring Challenge put forward by RSA Laboratories.