A Logic of Authentication

### 2020310795 康鸿博

## What are the problems mentioned by the paper? (intro)

In this paper, the authors point out a problem: it’s hard to design an authentication protocol that survive from different attacks, especially when people want to achieve better performance by reducing the data sent or calculated as much as possible. The difficulties exist in different aspects: (1) The initial data (e.g. whether you know the public key of another node, or you need to get that from a trusted server) and the basic assumptions (e.g. whether there’s a synchronized clock) hold and trusted by principals are different because of different network setting. (2) The protocol designer must prohibit “any” possible attack to the protocol, especially those not found clearly stated yet. This means that it’s inadequate for the protocol designer to use a “black list” approach by saying the protocol is protected from some attacks, but they need to generate a “white list” proof for the protocol.

The authors give concrete analysis for four commonly used authentication protocols based on its predicate logic, and show how common variations lead to potential attacks.

## Summary of major innovations (intro)

This is the very first paper to analyze authentication protocols using simple logic. It shows good results by proving correctness in some widely used protocols, and shows flaws in some of their variations. These flaws are also identified and fixed by sporadic literature, but this is the first paper to give a theoretical and systematic analysis of authentication protocols about correctness and succinctness.

## How about the important related works/papers? (related work)

Four widely-used authentication protocols are analyzed in the paper as example: Kerberos Protocol, Andrew Secure RPC Handshake, Needham-Schroeder Public-Key Protocol, CCITT X.509 Protocol. These algorithms have different assumptions and have different design targets:

1. The design target of Kerberos Protocol is similar to that of the “RSA key exchange” protocol: to build an encrypted message channel between two servers. Kerberos use a trusted third party, and it prohibit the man-in-the-middle attack.
2. The design target of Andrew Secure RPC Handshake is to get a new session key when two servers already have a key.
3. The design target of Needham-Schroeder Public-Key Protocol is to allow two servers, each with their own public key and private key, to exchange two secret number, under the assistance of a trusted server.
4. The design target of CCITT X.509 Protocol is to send Xa, Ya from A to B, and send Xb, Yb from B to A, ensuring the integrity of Xa and Xb, assuring the recipient of their origin, and guarantees the privacy of Ya and Yb.

## What are some intriguing aspects of the paper? (design & implementation)

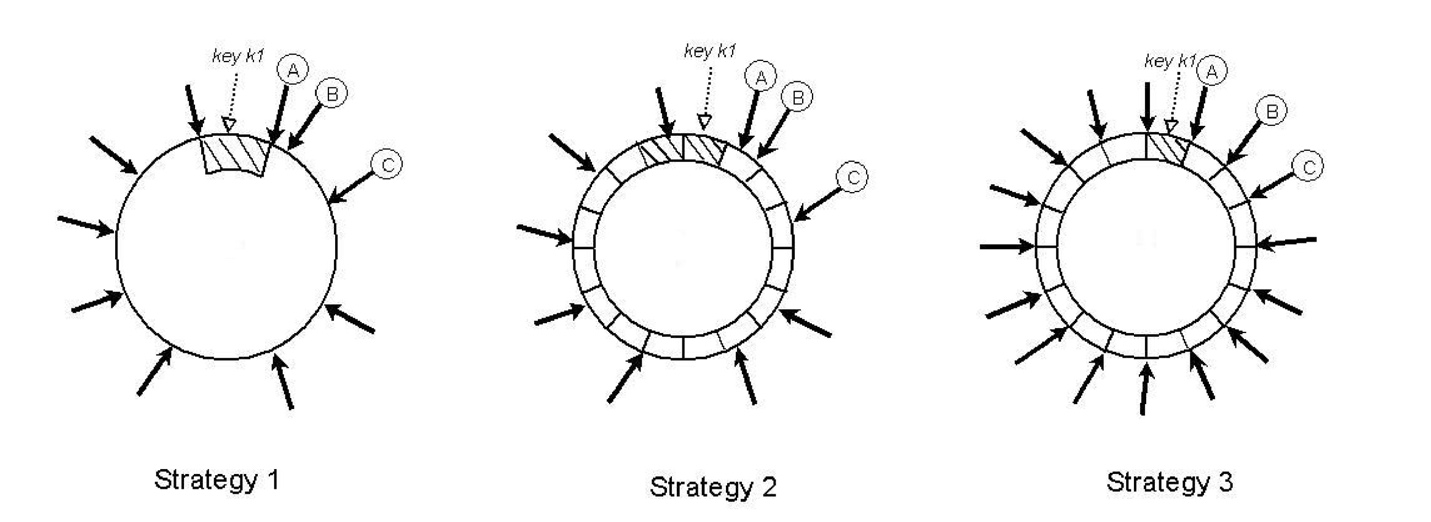
###### Predicates:

1. P believe X: P believe that X is true
2. P sees X: P receive message about X
3. P said X: P sent message about X
4. P controls X: P has an authority on X, this generally means that P is responsible for (the generation) of X.
5. Fresh(X): X is a fresh message from this execution of the protocol. This help prevent replay attack.

###### Consistent Hashing(Figure3):

As in the following figure, consistent hashing is a 2-step algorithm. For each (key, value) pair, it hashes the key into the hash space. Each key should belong to some range in the hash space controlled by one node. It then stores the pair to the that node.

In Dynamo, the total hash space is divided into Q subranges, and each subrange is assigned to some node randomly, which guarantee the load balance.



###### Replication:

To support high availability and durability, Dynamo replaces its data on multiple nodes. To be specific, each (key, value) pair is not only stored on the node holding its range, but also on nodes holding the following ranges. This enable load balance with replication, and support easy data retrieval on failures.

###### Data versioning:

Since Dynamo only support eventual consistency, it can exploit more parallelism by data versioning. Its versioning is similar to that in “causal consistent” systems, where each version is identified by an “update list”, containing timestamps for all relevant nodes. The difference between the consistency algorithm in Dynamo and that in “causal consistency” algorithms is that it allows nodes to reply to “get” requests with a value that may not be the “latest” in the system. This may be a result from node failure, or too many concurrent accesses. This brings the problem that multiple concurrent versions for the same key may exist in the system. Dynamo don’t solve this, but send all these versions, and let the conflict solvers provided by the program engineer solve this.

###### Quorum-based replication management:

For each get/put operation, traditional quorum-based algorithms send it to the N corresponding nodes, and wait until receiving R / W (constants) confirmations. In Dynamo, a weaker quorum is used, where W writing nodes don’t necessarily be in the first N nodes.

This allow “put” operations to succeed even if the “original node” in the quorum fails. The substitute nodes will then send the value back after the node recover.

Note that the substitute node may fail, so the original node will lose the update. Dynamo solves this by Merkle tree.

###### Membership and failure detection:

Membership is maintained by a list. Nodes periodically gossip to each other to synchronize the list (like in many network protocols). To support fast synchronization, some nodes are picked by the administrator as seeds, and all nodes try to sync their state with seeds. Node failures are detected by heartbeat messages. Adding / Removing nodes are done by calling some function.

## How to test/compare/analyze the results? (experiment)

The author report results generated by the execution of Amazon products. It shows that the system reaches good load balance, especially when under high load. It also shows that system designers shouldn’t just focus on the average latency, because the

99.9% latency can be orders of magnitude higher than the average latency. Focusing only on average latency may lead to bad user experience.

## How can the research be improved? (the bad side, future work, your idea)

For my own idea (or future research interest), I’m interested in: (1) how can we apply small “transaction” if needed? (2) how can we utilize the “heterogeneity” in the network? Since message passing in datacenter is much faster than those between datacenters.

## If you write this paper, then how would you do? (your idea)

I like this paper very much, and I don’t think I can do better than it. I think it’s a good system paper, which has brave hypothesis from real world (instead of theoretical analysis), clear paper structure, and long-term real-world data.

## What’s your test Results about the paper? (your action)

Nope.

## Give the survey paper list in the same research area (your survey)

[1] [Fay Chang](https://dblp.org/pid/13/6463.html), [Jeffrey Dean](https://dblp.org/pid/d/JeffreyDean.html), [Sanjay Ghemawat](https://dblp.org/pid/g/SGhemawat.html), [Wilson C. Hsieh](https://dblp.org/pid/31/1764.html), [Deborah A. Wallach](https://dblp.org/pid/53/6138.html), [Michael Burrows](https://dblp.org/pid/73/615.html), [Tushar Chandra](https://dblp.org/pid/26/477.html), [Andrew Fikes](https://dblp.org/pid/71/5497.html), [Robert Gruber](https://dblp.org/pid/g/RobertGruber.html):  
**Bigtable: A Distributed Storage System for Structured Data (Awarded Best Paper!).** [OSDI 2006](https://dblp.org/db/conf/osdi/osdi2006.html#ChangDGHWBCFG06): 205-218

[2] [Venugopalan Ramasubramanian](https://dblp.org/pid/85/4250.html), [Emin Gün Sirer](https://dblp.org/pid/s/EminGunSirer.html):  
**Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays.**[NSDI 2004](https://dblp.org/db/conf/nsdi/nsdi2004.html#RamasubramanianS04): 99-112

[3] [Antony I. T. Rowstron](https://dblp.org/pid/r/AITRowstron.html), [Peter Druschel](https://dblp.org/pid/d/PDruschel.html):  
**Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility.**[SOSP 2001](https://dblp.org/db/conf/sosp/sosp2001.html#RowstronD01): 188-201

[4] Lu, Y., Shu, J., Chen, Y., & Li, T. (2017). Octopus: an rdma-enabled distributed persistent memory file system. In 2017*USENIX Annual Technical Conference*(USENIX ATC 17) (pp. 773-785).