Tatra Mt. Math. Publ. **33** (2006), 31–40

Tatra Mountains
Mathematical Publications

DISTRIBUTED TIME-STAMPING WITH BOOMERANG ONIONS

Marcin Gogolewski — Mirosław Kutyłowski — Tomasz Łuczak

ABSTRACT. We propose a distributed time-stamping scheme in a system with anonymous communication. Our scheme, based on onion protocol and random witnesses technique, is fully distributed with a relatively low communication overhead. It preserves anonymity against an adversary who can trace a constant fraction of communication lines.

1. Introduction

Time-stamping. Time-stamping is a service of confirming existence of an electronic document at the time when a time-stamp was issued. Time-stamps are crucial for many applications, especially in electronic documents flow and for legal issues.

Quite often time-stamps are generated by tamper-resistant black-box devices (or at least tamper-evident devices) equipped with a high precision clock. However, a party that gains malicious control over time-stamping devices (for instance using kleptographic code [14]) may, over time, gain control in the legal system.

A different approach is to base time-stamps on a distributed system with logical clocks. It is a low cost solution, resistant to malicious hardware producers and scalable. It may offer detection of malicious Byzantine servers cheating about time. However, a serious drawback might be the lack of anonymity of service——this is a potential source of DoS attacks against certain parties involved in time-stamping.

Linking. Usual solution, better from a security point of view, is based on a technique known as linking. It is storage consuming and insecure in some situations—and for this reason rather impractical (see also [3, 1, 4]). A basic

---

2000 Mathematics Subject Classification: 68M14.
Keywords: anonymity, onion protocol, time-stamping.

Partially supported by KBN grants 0 T00A 003 23 and 2 P03A 016 23 in years 2002–2004.

version of this protocol looks as follows: as soon as server $\mathcal{S}$ obtains the nth data $\mathcal{D}_n$ to be time-stamped, it computes $d_n = H(\mathcal{D}_n)$, $C = (n, d_n, L_n)$, where H denotes a secure hash function, $L_n = \big(d_{n-1}, H(L_{n-1})\big)$, and d_{n-1}, L_{n-1} are values computed in a similar way for the previous data D_{n-1}. As a reply $\mathcal{S}$ sends C signed digitally. The idea is that each time-stamp is tied with its predecessor and if the server executes the protocol correctly, each time-stamp, except the last one, has exactly one successor. A forgery requires forking of the chain (or breaking the hash function scheme). Such forking might be detected and used as an evidence of forgery.

Random witnesses. Random witness technique yields time-stamps from authorities chosen in a pseudo-random way. Time-stamp for a document D must be obtained from k servers determined uniquely from $H(D)$. Since hash function H has a "random" behavior, the user is unlikely to work with corrupted (or allied) servers only. There is a major concern about privacy of this solution; to check whether D was time-stamped one may trace the traffic from the potential holders of D to k time-stamping authorities. Time-stamping authorities know who requested the time-stamp corresponding to D.

Problem statement and the main result. Random witness technique can be used together with an onion protocol—in this way each request is directed to the time stamping authority via an anonymous path. In this note we propose a slightly different, simpler solution. The user sends a single *boomerang* onion through the network: its source and destination are the same, and the servers on the route of the onion insert time-stamps into the onion processed. Even if communication delays occur, one receives the consecutive time-stamps with time differences matching plausible communication delays. Essentially, no specialized time-stamping servers are required. The servers on the route of the onion do not know the source (and destination) of the request, so denial of service against certain users is very difficult (which may be crucial for instance in the case of time-stamping stock market communication). Also tracing user activities is very hard.

Paper organization. In Section 2 we briefly describe some tools that may be used to achieve anonymity, in Section 3 we introduce *Onion time stamping protocol* which anonymity features are discussed in Section 4.

2. Anonymity tools

2.1 Mix servers

A mix [5] works as follows: a number of encoded messages enters a mix, they are recoded cryptographically by the mix and leave it. Recoding process guarantees that an external observer cannot link the incoming and the outgoing messages.

Usually, the messages are processed by many mixes forming a network (see, for instance [7]). This provides more security, since some of the mixes might be corrupted. Another point is that in a large system the capacity of a mix might be too low to process all messages simultaneously. In this case many mixes applied in parallel overcome the processing bottleneck.

2.2 Onions

An idea of mix servers is used for anonymous communication with messages encoded as *onions* [12, 11, 13]. The main idea is that each message is sent along a random path, but in this case the route of each message is determined by the sender. The messages are encapsulated cryptographically in a structure resembling an onion. We describe some details in the next subsection.

2.3 Onion protocol and traffic analysis

Onion protocol [12] (rediscovered in [13, 11]) is based on the Chaum's idea of mixes. Originally a fully connected network with n servers is considered. We assume that the whole communication between the servers can be eavesdropped; moreover, an adversary can get internal data from any constant fraction of servers.

According to the onion protocol, if a server A is to send a message m to a server B, then A chooses a random path $C_1, C_2, \ldots, C_\lambda = B$ for processing the message from A to B (λ is a security parameter discussed in [12], [8] and [10]). The message is encoded as an *onion*. It is encoded with an asymmetric encryption scheme so that when a server C_i obtains a *subonion* of the original onion, it *peels the subonion off* with its own private key, recovers the name of C_{i+1} and the subonion to be sent there, and finally sends the subonion to C_{i+1}. Encryption guarantees that no further knowledge is gained by C_i. Some more details of the onion protocol will be presented together with our protocol.

Let $E_K(x)$ denote x encrypted with a (public) key K and let $P(C_i)$ denote the public key of server C_i. Then a message M_1 encoding m leaving A has the form

$$E_{P(C_1)}(C_2, \ldots, M_2),$$

where M_2 will be described below. When server C_1 receives M_1, then, with the private key, it retrieves the name of the next server C_2 on the path and the encrypted packet M_2. Then C_1 sends M_2 to C_2. The message M_2 has a similar structure as M_1, namely

$$M_2 = E_{P(C_2)}(C_3, M_3).$$

More generally, for $i \leqslant \lambda - 2$,

$$M_i = E_{P(C_i)}(C_{i+1}, \ldots, M_{i+1}) \quad \text{for } i \leqslant \lambda - 2, \quad \text{and} \quad M_\lambda = E_{P(C_\lambda)}(\ldots, m).$$

So each server on the path gets a packet from which it can retrieve only the name of the next server on the path and the packet that should be forwarded there. The structure of the message resembles an onion: at each server one of its layers is peeled off, until the kernel is delivered to the final destination.

In [12] a modified version of the onion protocol is investigated. It was shown that then an eavesdropper cannot gain much information about packet destinations as far as λ is polylogarithmic in n. The parameter λ can even be reduced to $O(\log^2 n)$, as claimed in [6]. If an adversary can eavesdrop only a certain fraction of the communication lines [2], then $\lambda = O(\log n)$ already provides anonymity [10].

3. Onion time stamping protocol

The (onion) protocol is executed by servers $\mathcal{S}_1, \ldots, \mathcal{S}_n$ with communication lines described by edges of a graph G. The servers communicate by sending messages; each server can send messages only to its neighbors in G. We assume that each server $\mathcal{S}_i$ has a pair of keys for an asymmetric encryption; the public key is denoted by k_i. At each step a server receives onions, processes them, and sends the result to the next destination(s). It may also generate new onions or stop processing an onion if it is already at the destination point. Additionally all possible ciphertexts of asymmetric encryption should be equally distributed over some interval (e.g., in the case of RSA encryption $\left(1, \max_{i \in (1,n)} (m_i)\right)$, where m_i is modulus used in encryption by $\mathcal{S}_i$; for more details see [9]), otherwise an adversary can get some additional information.

For the sake of simplicity of analysis we assume that the servers work in rounds: during a single round a server receives messages from the other servers, makes some internal operation, and finally sends some number of messages to the other servers. However, one can easily see that our argument applies also to an asynchronous case.

Creating boomerang onions. We describe the protocol from the point of view of a user who fetches time-stamps from the system. We assume that there are many onions processed in the system, or some of the communication links cannot be traced by an adversary.

Let $E_k^{\mathrm{Sym}}(X)$ denote a hybrid ciphertext of X: a random key S is chosen; S is encrypted with the public key k; X is encrypted using a symmetric algorithm and key S in CBC mode; the ciphertexts of S and X are concatenated.

Assume that Alice wants a *time-stamp* for a document $\mathcal{D}$. Let H be a secure hash function and symbol $|$ denotes concatenation. Then Alice performs the following steps:

1. She computes $\mathcal{H}_j = H(\mathcal{D}|j)$, for $j \in \{1, 2, \ldots \lambda\}$.
2. She chooses a sequence of servers: $\mathcal{S}_{i_1}, \mathcal{S}_{i_2}, \ldots, \mathcal{S}_{i_{\lambda-1}}$; where $\mathcal{S}_{i_1} = \mathcal{S}_{i_\lambda}$ is her own server. This path will be used to construct a *boomerang onion* that has the same source and destination. The servers are chosen so that $\mathcal{S}_{i+1}$ is a neighbor of $\mathcal{S}_i$ in G, the first $\lambda/2$ servers are chosen in a random walk in G, where for each node of G all outgoing edges are chosen with the same probability. The remaining sequence of $\lambda/2$ servers is chosen in some arbitrary random way in order to get from $\mathcal{S}_{\lambda/2}$ back to $\mathcal{S}_{i_1}$.
3. Alice creates λ pairs of asymmetric keys (p_j, P_j) for $j \in \{1, \ldots, \lambda\}$, where p_j is a public key to be used by server $\mathcal{S}_{i_j}$, and P_j is the corresponding private key.
4. She chooses uniformly at random some numbers $r_1, r_2, \ldots, r_\lambda$.
5. Alice creates the *kernel* of the *forward onion*:
$$O_{F_\lambda} = E^{\mathrm{Sym}}_{k_{i_\lambda}}(\mathrm{ID}, r_\lambda, \mathrm{length}, \mathcal{H}_\lambda, \mathrm{Alice}, p_\lambda, \mathrm{PADDING}_0)\,,$$
where number length determines the length of a random string PADDING$_0$, which extends the data to some number of full blocks for symmetric block encryption.
6. Alice chooses at random a bit string L of the same length as O_{F_λ}, and defines the kernel of *acknowledgment onion* as $O_{A_\lambda} = L$.
7. Alice creates onions O_{F_j} for $j \leqslant \lambda$ by the following recursive formula:
$$O_{F_j} = E^{\mathrm{Sym}}_{k_{i_j}}(\mathrm{ID}, r_j, \mathrm{length}_j, \mathcal{H}_j, \mathrm{next}_j, p_j, \mathrm{PADDING}_j, O_{F_{j+1}})\,,$$
where ID is a fixed tag identifying the protocol, r_j is a fixed size random number, included in order to secure protocol against some attacks, next$_j$ is the number of the next server on the onion path, length$_j$ determines position of string, PADDING$_j$ it is used so that the onion $O_{F_{j+1}}$ starts at the beginning of a block of the symmetric encryption algorithm. In fact, since each onion must have the same length, we alter the definition of O_{F_j}: before encryption we truncate $O_{F_{j+1}}$ by some number of blocks (on the right hand side of it). In this way we loose some part of $O_{F_{j+1}}$, but it corresponds to a part of PADDING$_0$, so no useful data is lost. Finally Alice sends the onion O_{F_1} and O_{A_1} to the server $\mathcal{S}_{i_1}$. For $j < j'$ onion O_{F_j} is a *subonion* of $O_{F_{j'}}$.

Processing the onions. During a round server $\mathcal{S}_l$ receives a number of messages containing forward and acknowledgement onions. When a message (O_F, O_A) is received by a server, the server tries to decrypt the message with its own private key. If
$$O_F = E^{\mathrm{Sym}}_{k_{i_l}}(\mathrm{ID}, r, \mathrm{length}, \mathcal{H}, \mathrm{next}, p, \mathrm{PADDING}, O)\,,$$

then server $\mathcal{S}_l$ does the following:

1. extracts strings length, $\mathcal{H}$, next, p and O;
2. constructs O'_A in the following way:
   - it truncates O_A to make room for adding a *time-stamp*;
   - for a signing function σ computes
     $$s := \sigma_{K_l}(\mathcal{H} \,|\, \text{time} \,|H(O_A)|H(O_F)) ,$$
     where K_l is the private signature key of the server, *time* is the current time;
   - adds s at the beginning of O_A and encrypts the string obtained using key p;

   The number of bits truncated is chosen such that O_A and O'_A have the same length.
3. constructs O'_F as O with some random bit string added at the end of O, such that O_F and O'_F have the same length;
4. sends the pair (O'_F, O'_A) to server $\mathcal{S}_{\text{next}}$.

Finally, $\mathcal{S}_\lambda$ delivers the acknowledgment onion to Alice, as indicated in the kernel of the corresponding forward onion.

4. Protocol security

An acknowledgment onion received by a user contains encoded time-stamps of all servers on the path traversed by this onion. So it serves also as evidence that all servers on the onion path have processed the forward onion. If one of the servers gives a time-stamp with an incorrect time, time-stamps may become non-consecutive, and it can be used as an evidence of forgery (or at least as an evidence of inaccuracy).

In a distributed time-stamping protocol, a malicious Time-Stamping Authority may deny service selectively against some users, reveal who is ordering time-stamping service, or misbehave against other Time-Stamping Authorities. In our protocol a Time-Stamping Authority knows only from which server it got a pair of onions and where they have to be forwarded after recoding. So it is hard to deny service selectively or to reveal information about users' activity based only on the information available from the protocol.

Still, we have to consider a passive adversary who monitors the traffic. Since all messages sent are encrypted the only information available for an eavesdropper is the number of messages sent along the tapped links at each moment. In this way an adversary may try to determine recipients of the time-stamps. The analysis can be facilitated by the fact that the boomerang onions return to the starting point.

One of the key issues in the classical onion protocols is to estimate the length λ of the paths sufficient for achieving unlinkability of the sources and destinations of the messages. Techniques contained in [2, 10, 8] may be used to estimate the size of λ. We shall discuss here how these techniques can be applied for the case where some onions are boomerang onions sent according to the protocol.

4.1 Adversary

According to the model introduced in [2], an adversary can trace a constant fraction of communication lines between the servers. Even if the adversary cannot read an encrypted message (as it happens for our protocol), he may determine the number of messages sent along each tapped line.

We assume that apart from the boomerang onions used for time-stamping activities, messages are exchanged in the system according to the regular onion protocol. For the sake of simplicity we assume also that all users send their onions simultaneously, and each onion path has the same length λ.

The purpose of the traffic analysis is to reveal who is ordering time-stamps. We assume that the adversary can trace how many messages are sent from each server and how many messages are finally received by each server. In particular, if a server either only sends or only receives messages it is evident that no time-stamp has been ordered from this server. However, we consider the situation of a heavy load, where each server sends and receives many messages.

The main goal is to show that analyzing the traffic induced by our protocol brings no real advantage for an adversary with high probability.

For this purpose we consider variation distance between probability distributions. Recall that variation distance $d_{TV}(\mu_1, \mu_2)$ between probability distributions μ_1, μ_2 defined over space X equals:

$$d_{TV}(\mu_1, \mu_2) = \sum_{x \in X} \left| \mu_1(x) - \mu_2(x) \right| .$$

We analyze the variation distance between probability distribution of a random variable describing linking between the messages sent and messages delivered, and the same random variable conditioned by the traffic information available to the adversary.

4.2 A priori information

In a paper [2], B e r m a n *et al.* consider a model in which final destinations of the messages are chosen according to some a priori distribution known to the adversary. We deal here with some particular a priori distribution that is not uniform (the source and the destination of boomerang onion is the same). One of the major results in [2] says that it suffices to split the protocol into the first and the second half, each lasting for $\lambda/2$ steps, and show that the probability distribution after $\lambda/2$ steps is almost uniform with high probability, then the adversary gains not much information through traffic analysis. More precisely, if $\Pi^{(T)}$ stands for a random variable denoting positions of the messages after T steps, P stands for a random variable with traffic information available to the adversary, then

$$I\big(\Pi^{(\lambda)}\colon P\big) \leqslant I\big(\Pi^{(\lambda/2)}\colon P\big)\,,$$

where $I(A:B)$ denote mutual information of random variables A and B. On the other hand, according to Lemma 3 from [2], mutual information is related to variation distance in the following way:

- if $\Pr\big[d_{TV}\big((\Pi^{(\lambda/2)}|P),\Pi^{(\lambda/2)}\big) > \alpha\big] < \alpha$, then $I\big(\Pi^{(\lambda/2)}\colon P\big) = O\big(\log(n!)\cdot\sqrt{\alpha}\,\big)$,
- if $I\big(\Pi^{(\lambda)}\colon P\big) < \delta$, then $\Pr\big[d_{TV}\big((\Pi^{(\lambda)}|P),\Pi^{(\lambda)}\big) > \gamma\big] < \gamma$ for $\gamma = O(\delta^{1/3})$.

So in order to show that

$$\Pr\left[d_{TV}\big((\Pi^{(\lambda)}\,|\,P),\Pi^{(\lambda)}\big) > \frac{1}{n}\right] < \frac{1}{n}$$

(which is our main goal) it suffices to show that

$$\Pr\big[d_{TV}\big((\Pi^{(\lambda/2)}\,|\,P),\Pi^{(\lambda/2)}\big) > \alpha\big] < \alpha$$

for $\alpha = 1/(n^8\log^2 n)$.

When performing traffic analysis for the first $\lambda/2$ nodes of the paths traversed by the onions it does not make any difference if we have the onions sent according to the regular onion protocol, or the boomerang onions of our protocol. Indeed, in both cases the paths are chosen at random. Hence the main result from [10] applies: for a certain $\lambda/2 = O(\log n)$ the variation distance is lower than α. By our previous observations this concludes the analysis.

4.3 Adversary controlling servers

In the case when a limited number of servers is malicious and cooperates with an adversary we have to rewrite slightly the anonymity proof from [10]. The core technical point in the proof is to construct the so called coupling that guarantees convergence between two stochastic processes. These processes describe routes of onions. For a path coupling proof it suffices to consider only the case when the second process is obtained from the first one by just transposing positions of two onions.

In a coupling proof in [10] two consecutive steps of the protocol are considered. After the first step the difference between two processes "migrates" into two random positions, say A and B. At the next step the onions from positions A and B are moved into some positions A' and B'. With a probability at least $1 - p$ the positions A, B, A' and B' form a cross-over structure—that mean the adversary does not monitor any of the links: from A to A', from A to B', from B to A', and from B to B'. In such a case we can build dependencies between process one and process two, so that they reach the same state.

In the case of malicious servers probability of successful coupling decreases, but it is not lower than $1 - p - q$ where q is the probability that one of the servers A, B, A', B' is malicious. For instance, if only one server in the system is malicious, then $q = 4/n$, $1 - p - q$ is still bigger than some constant, and so essentially the same convergence rate is achieved. It is also easy to see that a larger number of malicious servers does not affect the convergence rate significantly.

5. Final remarks and conclusions

The protocol described in this paper provides anonymity of a time-stamping service. It is designed for distributed environments; in big networks with many messages it guarantees that even if some servers are overrun by an adversary, anonymity is assured with high probability. A user who requests a time-stamp needs to send only one boomerang onion. Our time-stamping protocol is scalable and does not suffer from bottleneck problems. It can be also easily extended to provide additional functionalities (robustness against DoS attacks, tools to analyze network state and examining honesty of some time-stamping servers).

REFERENCES

[1] ANSPER, A.—BULDAS, A.—SAAREPERA, M.—WILLEMSON, J.: *Improving the availability of time-stamping services*, in: 6th Australasian Conference on Information Security and Privacy, Lecture Notes in Comput. Sci., Vol. 2119, Springer-Verlag, Berlin, 2001, pp. 360–375.

[2] BERMAN, R.—FIAT, A.—TA-SHMA, A.: *Provable unlinkability against traffic analysis*, in: Lecture Notes in Comput. Sci, Vol. 3110, Springer-Verlag, Berlin, 2004, pp. 266–280.

[3] BULDAS, A.—LAUD, P.—LIPMAA, H.—VILLEMSON, J.: *Time-stamping with binary linking schemes*, in: CRYPTO '98, Lecture Notes in Comput. Sci., Vol. 1462, Springer--Verlag, Berlin, 1998, pp. 486–501.

[4] BULDAS, A.—LAUD, P.: *New linking schemes for digital time-stamping*, in: ICISC '98, Korea Inst. of Information Security and Cryptology, Seoul, 1998, pp. 3–14.

[5] CHAUM, D.: *Untraceable electronic mail, return addresses, and digital pseudonyms*, Communications of the ACM **24** (1981), 84–88.

[6] CZUMAJ, A.—KANAREK, P.—KUTYŁOWSKI, M.—LORYŚ, K.: *Distributed stochastic processes for generating random permutations*, in: ACM-SIAM SODA '99, 1999, pp. 271–280.

[7] DINGLEDINE, R.—SHMATIKOV, V.—SYVERSON, P.: *Synchronous batching: from cascades to free routes*, in: Workshop on Privacy Enhancing Technologies (PET) '04, Lecture Notes in Comput. Sci., Vol. 3424, Springer-Verlag, Berlin, 2004, pp. 186–206.

[8] GOGOLEWSKI, M.—KUTYŁOWSKI, M.—ŁUCZAK, T.: *Mobile mixing*, Lecture Notes in Comput. Sci., Vol. 3506, Springer-Verlag, Berlin, pp. 380–393.

[9] GOGOLEWSKI, M.—KUTYŁOWSKI, M.: *Secure data storing in a pool of vulnerable servers*, Artificial Intelligence and Security in Computing Systems, Kluwer Academic Publishers, pp. 217–226.

[10] GOMUŁKIEWICZ, M., KLONOWSKI, M., KUTYŁOWSKI, M.: *Provable unlinkability against traffic analysis already after $\mathcal{O}(\log(n))$ steps!*, in: Information Security Conference (ISC) '04, Lecture Notes in Comput. Sci., Vol. 3225, Springer-Verlag, Berlin, 2004, pp. 354–366.

[11] GULCU, C.—TSUDIK, G.: *Mixing e-mail with Babel*, in: Symposium on Network and Distributed System Security, IEEE, 1996, pp. 2–16.

[12] RACKOFF, C., SIMON, D. R.: *Cryptographic defense against traffic analysis*, in: ACM STOC '93, 1993, pp. 672–681.

[13] SYVERSON, P. F.—GOLDSCHLAG, D.—REED, M.: *Hiding routing information*, in: Information Hiding '96, Lecture Notes in Comput. Sci., Vol. 1174, Springer-Verlag, Berlin, 1996, pp. 137–150.

[14] YOUNG, A.—YUNG, M.: *The dark side of black-box cryptography*, in: CRYPTO '96, Lecture Notes in Comput. Sci., Vol. 1109, Springer-Verlag, Berlin, 1996, pp. 89–103.

Received September 30, 2004

Marcin Gogolewski
Tomasz Łuczak
Faculty of Mathematics and Computer Science
Adam Mickiewicz University of Poznań
Umultowska 87
PL–61-614 Poznań
POLAND

E-mail: marcing@amu.edu.pl
tomasz@amu.edu.pl

Mirosław Kutyłowski
Institute of Mathematics
Wrocław University of Technology
ul. Wybrzeże Wyspiańskiego 27
PL–50-370 Wrocław
POLAND

E-mail: Miroslaw.Kutylowski@pwr.wroc.pl