Anonymous IP-address Blocking in Tor with Trusted Computing (Short Paper: Work in Progress)*

Patrick P. Tsang[†], Apu Kapadia[†‡], Sean W. Smith[†]

[†]Department of Computer Science
Dartmouth College
Hanover, NH, USA

[‡]Institute for Security Technology Studies
Dartmouth College
Hanover, NH, USA

{patrick, akapadia, sws}@cs.dartmouth.edu

Abstract

How does one block an *anonymous* user hiding behind an anonymous routing network? In this paper, we outline a security protocol that uses resource-constrained trusted hardware to facilitate anonymous IP-address blocking in anonymizing networks such as Tor. Tor allows users to access Internet services privately by using a series of Tor routers to obfuscate the route from the client to the server, thereby hiding the client's IP address from the server. The success of Tor, however, has been limited because of malicious users who misuse the network. For example, anonymous users can deface websites or create malicious entries on websites such as Wikipedia.[1] Administrators of these websites routinely rely on IP-address blocking for disabling misbehaving users' accesses. The IP-address anonymity provided by Tor, however, makes it difficult for administrators to deny access to such offenders. As a result, administrators resort to blocking *all* Tor exit nodes, effectively denying anonymous access for all Tor's users. Our solution makes use of trusted hardware and allows services like Tor to provide anonymous blocking of IP addresses while requiring only a modest amount of storage at the trusted node.

*This research was supported in part by the NSF, under grant CNS-0524695, and the Bureau of Justice Assistance, under grant 2005-DD-BX-1091. The views and conclusions do not necessarily reflect the views of the sponsors.

[1]Wikipedia. `http://www.wikipedia.com`

1 Introduction

Anonymizing networks such as Crowds [8] and Tor [3] re-route a user's traffic between several nodes in different domains. Since these nodes are operated independently, users are able to trust the anonymizing network to provide anonymity. Real-world deployments of anonymizing networks, however, have had limited success because of their misuse. Administrators of websites are unable to blacklist malicious users' IP addresses because of their anonymity. Left with no other choice, these administrators opt to blacklist the entire anonymizing network. This approach eliminates malicious activity through such networks, but at the cost of the anonymity of honest users. In other words, a few "bad apples" can spoil the fun for everybody else using the anonymizing network. (In fact, this has happened repeatedly with Tor.[2])

To solve this problem, we present a secure protocol based on trusted hardware that allows servers to block *anonymous* users without knowledge of their actual IP addresses. Although this work applies to anonymizing networks in general, we consider Tor for purposes of exhibition. Building and prototyping a system based on our proposed solution is ongoing work. In this paper we present our proposed solution and protocol.

At a high level, our system provides users with an ordered collection of pseudonyms for each website. Each pseudonym is valid for a small time period, and a user's anonymity is maintained between these time periods. Without a *trapdoor*, the pseudonyms look unrelated to the website, and users enjoy anonymity

[2]The *Abuse FAQ for Tor Server Operators* lists several such examples at http://tor.eff.org/faq-abuse.html.en.

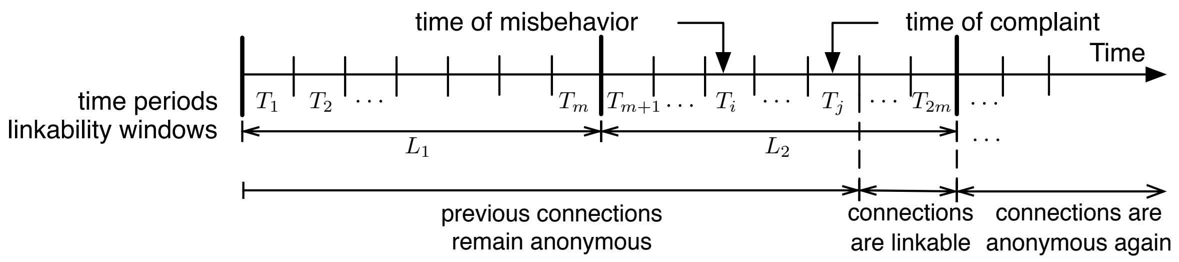
time of misbehavior
time of complaint
Time
time periods
linkability windows
T_1 T_2 ... T_m T_{m+1} ... T_i ... T_j ... T_{2m} ...
L_1
L_2
...
previous connections remain anonymous
connections are linkable
connections are anonymous again

Figure 1: The life cycle of a misbehaving user in our system

through the use of these unlinkable pseudonyms. Websites can, however, interact with a trusted node in the Tor network and obtain a trapdoor for a given connection, allowing the website to link a misbehaving user's *future* accesses. Our system thus provides "on-demand forward linkability" of a user's pseudonyms. Since users can verify whether they are on the website's "blacklist" *before* connecting to a website, blacklisted users can decide not to connect if they will be linked. The anonymity of users not on the blacklist remains unaffected. Since users need to be assured that they are checking the most recent version of the website's blacklist, the trusted node maintains the current version numbers of each website's blacklist. This requires only a moderate amount of storage at the trusted node, which is linear in the number of websites requesting the anonymous IP-blocking service.

Our solution makes use of trusted hardware as a Trusted Third Party (TTP). The TTP needs to be able to store its data and carry out computations without an external adversary—even the operator of the machine—being able to observe or manipulate it (except for destroying it altogether). To achieve this, we plan on using trusted hardware with the ability to provide a secure execution environment bound to private keys. In our prototype work, we are using an IBM 4758 [9] and plan to use an IBM 4764; however, we anticipate that similar environments will be enabled, perhaps with a lower degree of tamper protection, by architectures based on TPMs or newer CPU models such as XOM [6], AEGIS [10], or LT [4], or even a new type of TPM [1].

Our system has the following properties: (1) A user can check whether his or her future connections to a website will be linked. (2) A user who misbehaved at one website still enjoys anonymity with respect to other websites that have not complained about that user. (3) Connections made by a misbehaving user before the website complains remain unlinkable even though future connections can then be linked. (4) There is a period of time called the *linkability window* after which the misbehavior of users will be forgiven, effectively meaning that the users enjoy anonymity again.

Many in the community worry that trusted computing will be a vehicle for suppressing individual rights. We note that this project moves in the other direction, by using it to preserve privacy, and by using it to increase mainstream acceptance of privacy-enhancing technology such as Tor.

2 Our Solution

In our system, time is divided into linkability windows of duration L, each of which is split into m smaller time periods of duration T, as illustrated in Figure 1. We will refer to time periods and linkability windows chronologically as T_1, T_2, ... and L_1, L_2, ... respectively. While a user's access *within* a time period is tied to a single pseudonym, the use of different pseudonyms *across* time periods grants the user anonymity between time periods. Therefore, smaller time periods provide users with enough pseudonyms to simulate anonymous access. For example, T can be set to 5 minutes, and L to 1 day. If a user misbehaves, then the website may link any future connection from this user within the same day (the current linkability window). Consider Figure 1 as an example: A user misbehaves in a connection to a website during time period T_i within linkability window L_2. The website complains in time period T_j about that connection within the same linkability window. The website is then able to link future connections by the user in time periods $T_{j+1}, T_{j+2}, \ldots, T_{2m}$. Therefore, users cannot misbehave again for the rest of the day (the linkability window) once the website has detected the malicious behavior.

Users need to acquire a *token* from the TTP to connect to a website. The token contains the user's pseudonym for a particular time period and a *trapdoor* encrypted under the TTP's public key. When a website complains about a connection by presenting the token to the TTP, the TTP recovers the trapdoor from the token and returns it to the website. Given the trapdoor, the website can generate all the pseudonyms for future time periods within the current linkability window and maintain a blacklist with these pseudonyms. Users may query websites for their blacklists so that they know if they are being linked before they decide to establish connections to those websites. The authenticity, integrity and freshness of these blacklists are important to the security of our system. The TTP plays a role here by maintaining a bulletin of blacklist version numbers for public access. The version number for a website is updated every time a trapdoor is issued to that website.

Figure 2 illustrates how trapdoors and pseudonyms are generated. The process uses a novel hash-chain-like primitive first proposed by Ohkubo et al. [7] for securing RFID tags by ensuring both indistinguishability and forward security of the tags. We note that although the primitive we use in this paper shares similarities with that in [7], it possesses different security requirements that must be satisfied to secure our system as a whole. Trapdoors "evolve" throughout a linkability window through a *trapdoor-evolution function* H_1. Specifically, the trapdoor for the next time period can be computed by applying H_1 to the trapdoor for the current time period. A pseudonym is evaluated by applying the *pseudonym-evaluation function* H_2 to its corresponding trapdoor.

$$\begin{tikzcd}
\mathsf{seed} \arrow[r, "H_1"] & \mathsf{trpdr}_1 \arrow[r, "H_1"] \arrow[d, "H_2"] & \mathsf{trpdr}_2 \arrow[r, "H_1"] \arrow[d, "H_2"] & \mathsf{trpdr}_3 \cdots \arrow[d, "H_2"] & \mathsf{trpdr}_m \arrow[d, "H_2"] \\
 & \mathsf{nym}_1 & \mathsf{nym}_2 & \mathsf{nym}_3 \ldots & \mathsf{nym}_m
\end{tikzcd}$$

Figure 2: Evolution of trapdoors and pseudonyms

The TTP *seeds* the sequence of trapdoors (and hence the pseudonyms) with its secret, the source and destination IP addresses and the linkability window of the requested connection. Seeds are therefore specific to source-destination-window combinations. As a consequence, a trapdoor is useful only for a particular website to link a particular user during a particular linkability window. Since H_1 and H_2 are public knowledge, given trpdr_i, a website can compute $\mathsf{nym}_i, \mathsf{nym}_{i+1}, \ldots, \mathsf{nym}_m$.

We assume the use of a secure (encrypted and authenticated) communication channel between the TTP and other entities including the users and the websites. This can be realized by deploying Public Key Infrastructure (PKI) such as X509 at the TTP. SSL/TLS over HTTP is a natural candidate in the case of providing anonymous IP-address Blocking in Tor.

2.1 Security Requirements

Due to page limitation, we only highlight the security requirements of our system at an intuitive level instead of providing a rigorous security analysis, which we plan to do in our next paper.

1. *(Indistinguishability.)* No coalition of websites can link a user's connections within a particular linkability window unless at least one website in the coalition complains about that user within the linkability window.

2. *(Forward Linkability.)* A website can always link future connections made by a user within the same linkability window upon complaining about that user.

3. *(Backward Unlinkability.)* No coalition of websites can link a user's connections prior to complaining about that user.

4. *(Knowledgeability.)* A user can always tell correctly if a website has complained about him or her.

2.2 The System

We now present the algorithms and protocols executed by various entities in our system.

Setting up the System The TTP sets up the system by initializing itself with choices of various parameters such as which cryptographic algorithms to use, as well as generating all the necessary keys. In particular, the TTP generates the master secret key msk, a private-key and public-key pair (x, y) for a secure digital signature scheme $\mathcal{S}$ and a secret key k for a secure symmetric encryption scheme $\mathcal{E}$. It picks two distinct cryptographic hash functions H_1 and H_2. It also decides the duration L of a linkability window and the duration T of a time period such that $L = mT$ for some integer m.

Finally, the TTP publishes to the public the list of system parameters including $(\mathcal{S}, \mathcal{E}, y, H_1, H_2, L, T, m)$ and keeps to itself all secret information including (msk, x, k). The time is $t = T_1$ when the system starts.

Obtaining a Token User Alice (with IP address $\mathtt{IP}_U$) requests a token that can be used to connect to website W (with IP address $\mathtt{IP}_W$) during time period t. The TTP grants a request only if it is eligible,[3] in which case the TTP proceeds as follows. It first computes the seed, the trapdoor and the pseudonym for the source-destination-window tuple using key msk as follows:

$$\begin{aligned} \mathsf{seed} &\leftarrow H_1(\mathsf{msk}||\mathtt{IP}_U||\mathtt{IP}_W||\lfloor t/m \rfloor), \\ \mathsf{trpdr} &\leftarrow H_1^{(t \bmod m)}(\mathsf{seed}), \\ \mathsf{nym} &\leftarrow H_2(\mathsf{trpdr}). \end{aligned}$$

It then encrypts trpdr using $\mathcal{E}$ under key k, resulting in ciphertext ctxt, and signs the tuple $(t, \mathsf{ctxt}, \mathsf{nym}, \mathtt{IP}_W)$ using $\mathcal{S}$ under key x, resulting in signature σ. Finally the TTP sends back to Alice both the token $\mathsf{tok} = (t, \mathsf{ctxt}, \mathsf{nym}, \mathtt{IP}_W, \sigma)$ and the seed. Since the seed remains the same throughout the linkability window, Alice saves it securely only if it is the first time she obtains a token for a linkability window to a destination.

We note that allowing users to obtain tokens for future time periods in addition to the current time period has the following advantages: (1) the latency to establish a connection can be reduced by getting tokens prior to when they are needed. (2) The time of getting a token and that of using it are much less correlated, which makes some potential timing attacks a lot more difficult.[4]

Establishing a Connection Before establishing a connection to a website through Tor, a user first contacts the TTP for a token. As hinted earlier, this token contains encrypted information for later potential use by the TTP if the website complains about the user. This token can also be used by the website to link the connection being established to previous connections made by the same user if the website has complained against the user and obtained the necessary trapdoor. Therefore, before presenting this token to the website, the user must discern whether he or she will be linked with a previous connection. We now highlight the procedure for user Alice to establish a connection to website W.

1. *(Picking a Tor circuit.)* User Alice establishes a standard Tor connection to website W, using an entry node E, a middle node M, and an exit node X.

2. *(Getting the blacklist.)* Alice requests W's blacklist[5] from W through X. She also requests the current version number v of W's blacklist from the TTP through X. Before proceeding any further, Alice checks the freshness and integrity of the blacklist by verifying the TTP's signatures on the blacklist and makes sure the number of entries in the blacklist equals v.

   Now she checks if she is being linked by W by checking whether she is on the blacklist, which is indicated by the existence of an entry $\langle t_i, \mathsf{nym}_i, \sigma_i \rangle$ in the blacklist such that $H_2(H_1^{(t_i \bmod m)}(\mathsf{seed})) \stackrel{?}{=} \mathsf{nym}_i$. Alice continues to the next step if she is not being linked (or if she does not mind being linked).

3. *(Presenting the token.)* When Alice presents her token $\mathsf{tok} = (t, \mathsf{ctxt}, \mathsf{nym}, \mathtt{IP}_W, \sigma)$, website W checks if the token is a good one by verifying σ and the correctness of t and $\mathtt{IP}_W$. It then checks whether nym is in its blacklist, in which case W knows the connecting user is one of whom it complained against and may thus choose to decline the connection request. If W grants the request, it saves tok for a later potential complaint against the connection being established.

Complaining A website may complain about a connection during which the connecting user misbehaved, thereby enabling itself to link and thus block future connections made by the same user. In practice, there are scenarios when websites cannot decide quickly if a user misbehaved, e.g., before the connection is over, because the decision may involve relatively slow computation or human detection. Fortunately in our system, a website may complain about a connection at any time within the linkability window as long as the associated token is still present.

To complain about a connection, website W presents the TTP with the associated token $\mathsf{tok} = (t, \mathsf{ctxt}_t, \mathsf{nym}_t, \mathtt{IP}_W, \sigma_t)$. Assume the time now is $t'(\geq t)$. The TTP verifies signature σ_t, makes sure the IP address of W matches $\mathtt{IP}_W$ and $t'+1 \mod m$ is not zero[6] before it proceeds, in which case it does the following.

[3]For instance, t is no earlier than the current time period of the system and $\mathtt{IP}_U$ is indeed the IP address of U.

[4]As an example, the collusion of a corrupt system administrator of the TTP and exit nodes cannot directly correlate traffic of token requests at the TTP and the traffic of connection requests at the exit nodes to infer the identity of the connecting user.

[5]The first three columns in Figure 3.

[6]If $t'+1 \mod m = 0$, the system is at the last time period of the current linkability window and thus misbehaving users will be forgiven from the next time period anyway.

Linkable since	Signed nym	sig.	Current trapdoor	Current nym
t_1	$\mathsf{nym}_1^{(t_1)}$	$\sigma_1^{(t_1)}$	$\mathsf{trpdr}_1^{(t)}$	$\mathsf{nym}_1^{(t)}$
t_2	$\mathsf{nym}_2^{(t_2)}$	$\sigma_2^{(t_2)}$	$\mathsf{trpdr}_2^{(t)}$	$\mathsf{nym}_2^{(t)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Figure 3: The blacklist of a website. The first three columns are open for public query. The last two are secretly kept for the purpose of linking.

Website's IP	Version #	Timestamp
123.123.123.123	10	σ_1
222.222.222.222	6	σ_2
$\vdots$	$\vdots$	$\vdots$

Figure 4: The blacklist version-number bulletin

1. It decrypts the ciphertext ctxt_t to recover the trapdoor trpdr_t for time period t.
2. It computes the trapdoor $\mathsf{trpdr}_{t'+1}$ for time period $t'+1$ as $H_1^{(t'-t+1)}(\mathsf{trpdr}_t)$ and then the corresponding pseudonym $\mathsf{nym}_{t'+1}$ as $H_2(\mathsf{trpdr}_{t'+1})$.
3. It signs the pair $(t'+1, \mathsf{nym}_{t'+1})$ using $\mathcal{S}$ under key x, resulting in signature $\sigma_{t'+1}$.
4. It sends the tuple $(t'+1,\ \mathsf{nym}_{t'+1},\ \sigma_{t'+1},\ \mathsf{trpdr}_{t'+1},\ \mathsf{nym}_{t'+1})$ back to the website, after which the website adds the tuple as an entry to its blacklist.

Figure 3 shows the blacklist of a website.

At the same time, the TTP updates the blacklist version-number bulletin, which is a publicly readable database that holds the blacklist version numbers of all websites. A website's blacklist version number starts with 0 at the beginning of each linkability window and is equal to the number of entries in the website's blacklist. Therefore, every time a website W is given a new trapdoor, the TTP updates W's entry in the bulletin by incrementing the W's version number and re-timestamping the entry. See Figure 4 for an illustration of the bulletin.

Refreshing the Blacklist Different pseudonyms and thus trapdoors are required to link connections made by a misbehaving user at different time periods. In our system websites are capable of "evolving" trapdoors for a particular time period to those for any future time period within the same linkability window. In this way, websites need not ask the TTP for trapdoors to link a misbehaving user for every single time period.

At the beginning of every time period, websites refresh their blacklists to get themselves ready to link connections from misbehaved users. A website evolves the *current trapdoors* in the list by applying H_1 on them, i.e. $\mathsf{trpdr}_{t+1} \leftarrow H_1(\mathsf{trpdr}_t)$, and then computes the new *current nyms* from the newly evolved trapdoors, i.e. $\mathsf{nym}_{t+1} \leftarrow H_2(\mathsf{trpdr}_{t+1})$. See the last two columns of Figure 3.

Entering the Next Linkability Window Misbehavior is forgiven every time the system enters a new linkability window. Users who misbehaved previously can then connect to websites anonymously until they misbehave and are complained against again. Therefore, the tokens, trapdoors and pseudonyms which are specific to one linkability window become useless when the system enters a new window. Consequently, websites throw away tokens and empty their blacklists while the TTP resets all version numbers to zero at the end of every linkability window.

The duration m of the linkability window serves at least three useful purposes: 1) *(dynamism)* since IP addresses can get reassigned to different well-behaved users, it is undesirable to blacklist an IP address indefinitely, 2) *(efficiency)* the TTP must perform $O(m)$ hash operations to compute the trapdoor for each token request from a user; it is thus desirable to keep m bounded, and 3) *(forgiveness)* users can be forgiven for their misbehavior after a certain period of time.

3 Discussion and Future Work

We plan on prototyping our proposed system on an IBM 4758/4764 cryptographic coprocessor and evaluating its performance. In particular, we plan to evaluate how many connections the TTP can handle since it will be a central bottleneck for access to websites that require users to obtain tokens. As part of our ongoing work, we are continuing to revise the protocol and revisit design options and trade-offs. For example, is it possible to hide the IP addresses from the TTP when getting a token from it? Is it possible to reuse the token for multiple connections while preserving on-demand forward linkability? Is it useful to maintain different linkability windows for different websites? We now discuss the issues of trust and efficiency in our system.

Trust assumptions Properly configured, trusted computing platforms can provide a high degree of assurance that the sensitive information about a user's connection is visible only to the TTP. In our system, token generation and complaint handling by the TTP involves sensitive information such as source and destination IP addresses and various secret keys that must be secured.[7]

We posit that while users would be hesitant to trust a third-party machine administered by a fallible human operator with such information, they would be willing to be more trustful of a trusted third party machine running trusted hardware since sensitive data is insulated from the human operator. Any trusted hardware realizing the TTP in our system must therefore be capable of providing a secure execution environment that prevents sensitive information from leaking or being tampered with during computations. The TTP must also store its secret keys securely. Secure storage must therefore also be provided by the trusted hardware. In case a user Alice wants to be assured that the trusted hardware is still in a secure state before putting her trust on the TTP, the trusted hardware will need to be able to prove it. For example, IBM 4758 and 4764 have outbound authentication [9] while TCG TPMs can do remote attestation [1]. To reduce the amount of trust a user needs to place in the TTP, we are currently looking at ways to distribute this trust between several TTPs.

Securing the TTP's answer to blacklist version numbers can possibly be simplified by storing the information on a relatively less trusted database. This is possible because the information is publicly readable and thus requires no confidentiality, which makes protecting it with full-blown secure storage techniques unnecessary. We do, however, need to guarantee the authenticity, integrity and freshness of the version numbers. We are currently exploring techniques to reduce the storage requirements within trusted hardware.

Efficiency The high security assurance provided by trusted hardware makes it possible for us to provide anonymous IP-address blocking. A single TTP, however, limits the scalability of the system.[8] In our system, the computation by the TTP is linear in the number of connections made within the system. We plan to measure the amount of load a single IBM 4758/4764 coprocessor can handle. This will give us an idea of how many TTPs are needed in practice. We also plan on exploring the use of TPMs. For example, even though TPMs are more resource constrained and have lower assurance than tamper-resistant hardware such as the IBM 4758/4764, one can use current techniques such as thresholding [2], remote attestation [1], and private information retrieval [5] to reduce the amount of trusted information within the TPM.

For each user, the website needs to verify the authenticity of the user's token and check the user's pseudonym against its blacklist, which is a simple comparison. Refreshing the blacklist for the next time period is also efficient because it takes two hash operations per entry. Since a user needs to check whether its pseudonym is blacklisted, the user needs to download the website's blacklist. The size of the blacklist is linear in the number of blacklisted IP addresses within the linkability window, but we expect that the size of these blacklists will be negligible compared to the number of valid IP addresses.

[7]Two examples: (1) compromising the master secret key allows an adversary to produce unlinkable tokens; (2) a malicious system administrator monitoring the computation of the TTP knows the source and destination information of all connections.

[8]For instance, the IBM 4758 secure coprocessor is only as powerful as an Intel 486 machine with 4MB of RAM and 2MB of Flash.

4 Conclusion

We present a system based on trusted computing that provides *anonymous* IP-address blocking in anonymizing networks such as Tor. Users connect to websites using an ordered collection of seemingly unrelated pseudonyms. Websites can obtain trapdoors for a connection associated with a misbehaving user and link all the future pseudonyms for that user. Websites can therefore maintain blacklists to deny access to misbehaving users *without* knowing their IP addresses. By leveraging trusted hardware, we aim to provide a useful service to (behaving) users — our solution will enhance the acceptability of networks such as Tor to websites that are common targets of vandalism, thereby punishing misbehaving users while providing anonymity to the remaining users.

5 Acknowledgments

This paper greatly benefited from discussions with Sergey Bratus, Alexander Iliev, and Anna Shubina.

References

[1] Trusted computing group, May 2005. `https://www.trustedcomputinggroup.org/home`.

[2] Yvo G. Desmedt and Yair Frankel. Threshold cryptosystems. In *Proceedings on Advances*

in cryptology (CRYPTO '89), pages 307–315, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[3] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Usenix Security Symposium*, pages 303–320, August 2004.

[4] David Grawrock. Lagrande architecture. IDF Fall 2003 presentation, September 2003. `http://www.intel.com/technology/security/downloads/scms18-LT_arch.htm`.

[5] Alexander Iliev and Sean W. Smith. Private information storage with logarithmic-space secure hardware. In *3rd Working Conference on Privacy and Anonymity in Networked and Distributed Systems (I-NetSec '04)*, pages 201–216, Toulouse, France, August 2004. IFIP, Kluwer.

[6] David Lie, Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. Architectural support for copy and tamper resistant software. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pages 168–177, November 2000.

[7] Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. Cryptographic approach to "privacy-friendly" tags. In *RFID Privacy Workshop*, MIT, MA, USA, November 2003.

[8] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, November 1998.

[9] Sean W. Smith and Steve Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks (Special Issue on Computer Network Security.)*, 31:831–860, April 1999.

[10] G. Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing. In *Proc. of the 17th Annual ACM International Conference on Supercomputing*, pages 160–171, March 2003.