Securing Network Content

Diana Smetters Van Jacobson
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
{smetters,van}@parc.com

1 Introduction

The goal of the current Internet is to provide content of interest (Web pages, voice, video, *etc.*) to the users that need it. Access to that content is achieved using a communication model designed in terms of connections between hosts. This conflation of *what* content you want to access with *where* (on what host) that content resides extends throughout current network protocols, and determines the context in which they offer network security.

Trust in content – that it is the desired content, from the intended source, and unmodified in transit – is determined by *where* (from what host) and *how* (over what kind of connection) the content was retrieved. A user believes they are reading the news from the New York Times when they access it via a user-friendly name for the authoritative source of that news – `www.nytimes.com`; whether guessed, known, or obtained from a trusted directory (Google) in response to a few relevant search terms. Implicitly, they are also trusting that a) the DNS has given them a reliable indicator of where to find a host authorized to "speak for" the name they are interested in, b) they have actually made an HTTP connection to that host or its delegate (such as a content distribution network, or CDN) and c) that the content retrieved over that connection is unaltered by any unauthorized intermediary.[1]

A user accessing a higher-value resource, say their bank account, may do so with confidence only over a connection further authenticated by a cryptographic protocol such as TLS [10]. This gives them assurance that not only is the data coming from the intended source (as determined by the trust model for evaluating digital certificates built into their software client), and that it has not been tampered with in transit, but additionally that it may be protected from eavesdropping by encryption.

This connection-focused approach to security inexorably ties the security of content to trust in the host that stores it, and then additionally requires secure mechanisms to identify, locate and communicate with that host. This is widely recognized as a significant problem [11, 21]. Because the "trust" the user gets in the content they depend upon is tied inextricably to the connection over which it was retrieved, that trust is *transient* – leaving no reliable traces on the content after the connection ends, and cannot be leveraged by anyone other than the original retriever. A user who has stored a piece of content that they directly retrieved cannot be sure that it has not been modified since (*e.g.* by malware) without re-retrieving it prior to use. Another user interested in the same content cannot simply get it from the first; to have any confidence in that content they must retrieve it for themselves from the original source.

Going further, recent proposals [17, 5, 32, 6, 2, 19, 7, 21, 26, 31, 9, 3, 13, 20] suggest revising the very nature of networks to work natively in terms of content – in other words, the *name* a user would express to the network to retrieve a piece of content refers in some way to that content item itself, rather than to some specific host on which the content can be found. To be effective, such architectures require that not only can content be retrieved by name, but that the content so retrieved is *valid* – an authenticated, unmodified copy of the desired content produced by a publisher acceptable to the consumer.

Decoupling *what* from *where* requires security and network primitives that can refer to, and authenticate content itself, rather than the host and file containers where it resides. *Content-based*, rather than connection-based, security would allow users to securely retrieve desired content by name, and authenticate the result regardless of where it comes from – the original source, a copy on their local disk, or the user next to them in Starbucks™, or how it was obtained – via insecure HTTP, or new content-based mechanisms.

There have been a number of proposed mechanisms for

[1] A proxy server or a NAT box might make authorized modifications to that content.

securing named network content, focusing on making it possible to retrieve content based on *self-certifying names* – names constructed from the cryptographic digest of the content itself, or the key of the publisher used to digitally sign it [8, 16, 5, 32, 28, 21]. The advantage of such names is that they are globally unique, and can be generated by simple, completely autonomous local computation.

The two key disadvantages of such a scheme are: first, such opaque names form a flat, location-free namespace. This makes it extremely difficult to construct an efficient mechanism to retrieve a nearby copy of content corresponding to a particular name [7, 23], rather than resolving names using a location-independent mechanism such as a DHT [22, 30, 34].

Second, and more critical for the current discussion, users traffic in intuitive names. Even things like email addresses and hostnames are often impenetrable to them [2, 15, 14, 4]. Postulating a universe of flat, opaque names requires an indirection architecture akin to today's DNS to map from user-accessible names to network names [2, 6]. So the original content security problem still remains, it has simply been changed to the problem of securing this mapping. If the mapping can be subverted, the user ends up with the secure, self-verifying name for a wrong or even malicious piece of content.

To address this problem we authenticate the *linkage* between names – arbitrary names, including user friendly ones – and content rather than authenticating the content or its publisher. This allows us to securely retrieve that content from anyone who has it, regardless of what we choose for its name, or what mechanism – insecure or not – we use to resolve it to retrieve the corresponding content. This approach directly supports a number of previous proposals [8, 22, 28, 21], while providing greater flexibility, usability and security.

In the remainder of this paper we first motivate our approach, and contrast it with previous approaches to securing named network content. We then present an outline of how to make this work in practice, both in terms of low-level mechanisms for efficient verification of network content, and higher-level mechanisms for establishing trust in that content. Finally, we outline a number of near-term applications of securing content to enhance trust and efficiency on the Internet.

2 Authenticating Named Content

Secure content distribution requires that any *receiver* be able to reliably assess three properties of each piece of content received:

1. Its *validity*: is it a complete, uncorrupted copy of what the publisher sent.[2]
2. Its *provenance*: is the publisher one the receiver is willing to trust to supply this content.
3. Its *relevance*: is this content an answer to the question the receiver asked.

Current common network attacks (spam, pharming, phishing, cross-site scripting, man-in-the-middle, etc.) all rest on the receiver's inability to assess one or more of these properties. Unfortunately, most current proposals to name content on the Internet do little to dispel the receiver's darkness.

2.1 Self-Certifying Names

A common proposal for a content-oriented naming scheme for the Internet is to use *self-certifying names* – names for hosts [24] or content items [16] where the name itself is cryptographically constructed so that one can securely determine whether a given piece of content matches a given name. The simplest form of self-certification, *hash-verified* data, simply names a piece of content directly by its cryptographic (*e.g.,* `SHA-1`) digest [16, 8]. This allows the receiver to assess validity but conveys nothing about provenance or relevance. *Key-verified* data names a piece of content by the digest of the public key used to sign that data [8, 28]. Such names add some ability to assess provenance.

Both require a secure indirection mechanism to map from the name understood by users to the self-certifying name for a piece of content. Without such a mechanism, which must face all of the name contention problems that "semantic-free" names such as these were designed to sidestep [5, 32], a user can easily be deceived into accepting the wrong self-certifying name for the content they are interested in. The result is that they are indeed holding a "name" that directly authenticates a piece of content – only they have no idea that it was not the content they wanted. In other words, the architecture provides no way for the user to assess relevance.

Attempts to address this problem allow some user control over names. They allow a content publisher P to select a user-friendly tag *label* for a piece of content to be incorporated into the name along with P's public key. The resulting name of the content C might be $Digest(Pubkey_P||Label_C)$ [22], or $Digest(Pubkey_P)||Label_C$ [21], where $||$ denotes concatenation. The publisher P is expected to ensure that $Label_C$ is unique within the set of content items signed by $Pubkey_P$, so that the resulting name is itself unique.

[2] This combines both the traditional notion of *integrity*, that the data has not been altered, and *authenticity*, that this is the data corresponding to the name given by the user.

The content is authenticated by determining whether it was signed by $Pubkey_P$.

Unfortunately, both approaches are flawed – they do not allow users to verify whether a given piece of content is associated with a given intuitive name. In both, the content C is signed by P, but the label is not. Any piece of content signed by P can be undetectably substituted for any other with a different label, and an attacker can associate a new label of their choice with any content published by P.[3]

2.2 Authenticating the Link Between Names and Content

We would like to allow names to take any form while still allowing secure retrieval of content by name. We achieve this by the simple approach of authenticating not names, or content, but the *link* between them. What a publisher P "says" when they publish a piece of content is "N is my name for content C". We represent this by having P digitally sign the mapping from his chosen name N to C (possibly along with some additional metadata). That content is made available in the network as a mapping triple: $M_{(N,P,C)} = (N, C, Sign_P(N, C))$. A consumer can then usefully query for an arbitrary name N, and authenticate both the resulting content and its association with N, without having to know *a priori* either a unique, self-verifying identifier for C, or the specific identity of P. Clearly this allows any receiver to robustly assess validity, provenance and relevance. It also has a number of other advantages:

First, regardless of the form of N, this approach achieves *location-independence*. Because one can cryptographically verify the authenticity and correctness of both C and the relationship between N and C given such an $M_{(N,P,C)}$, one can obtain $M_{(N,P,C)}$ from anyone that has it – not just directly from P or from a location that might be specified in N.

Second, this approach allows the authentication of arbitrary forms of name N – opaque or semantically meaningful, self-verifying or not, unique or not, globally-understandable or encrypted, flat or structured. Naming systems are used to solve a huge variety of problems and there is no one "true name" for a piece of content. Some names are ontological, organizing the content into systems of meaning. Some are taxonomic, locating the content in an organizational, topological, representational, physical or other hierarchy. Some are simple distinguishers, differentiating an object in a set of similar objects. By allowing any name and securing the linkage between it and its intended referent, the value and meaning added by the naming system is preserved and secured. The navigation mechanisms used for resolving names to content may limit the form that names can take, but these mechanisms are themselves dependent on the particular collection of content, its semantics and intended use. Architecturally it is best to have an unrestricted mechanism and allow publishers and consumers to agree on what, if any, policies and constraints must be placed on their names.

Third, it is often easier for a consumer to recognize that P is a valid (to her) source for N, than to identify *a priori* a specific P whose version of N she wants. And it is the latter she must do if she wants to construct a query for data in a key-verified system where $N = Digest(P)$. For example, a consumer might trust content published by anyone working in her company, a status she can check for a specific P given an instance of a signed mapping $M_{(N,P,C)}$, without wanting to list all of her colleagues up-front in a query.

2.3 Having it All

There is a common belief – embodied most eloquently in Zooko's Triangle [33] that names can be simultaneously at most two of human-understandable, secure, and globally unique. Self-certifying names are secure and unique, `URLs` are human-understandable and unique.

We have chosen to simplify this problem by separating the roles of *identifier*, important to the human user in determining what content they want, and *locator* interpreted by the network in order to retrieve that content, from the security-critical *authenticator* used to determine whether the returned content is valid. Self-certifying naming schemes ask names to play both these roles simultaneously. We require names to play only the role of identifiers and locators, instead using auxiliary information – a digital signature – as our authenticator. In essence, we are *certifying* content, just as a digital (identity) certificate certifies the binding between a name and a public key. We have moved from a model where names must carry their own authentication information to one where we only require that receivers be able to authenticate the results they get back. This makes explicit the fact that it is only the receiver who can determine whether retrieved content is indeed valid, relevant, and has an acceptable provenance for their purpose.

This approach is essentially independent of the structure chosen for names, and leaves the problem of interpreting the name as a *locator* up to the underlying networking system in use. We can sign the mapping from a `URL` to a web page, and authenticate that content even if we retrieve it via insecure DNS and HTTP, or get it from an untrusted cache. We can achieve the same for content-based networking systems that route directly on names to

[3] Unless $Label_C$ is, say, the digest of C (in which case it is no longer user-determinable or user-friendly).

retrieve content [20].

This separation becomes even more important when we realize that it is only the locator function of names – the ability to use them to figure out where to find the desired content – that imposes a requirement that names be globally unique. When we authenticate the mapping from names to content that mapping – $M_{(N,P,C)}$ – is *always* unique, so the content consumer can always achieve functional uniqueness – distinguishing P_1's version of N from P_2's. If P_1 publishes multiple different pieces of content under the same name N, these can also be differentiated by their signatures.[4] (Though that content might be more useful to the consumer if we augment the information P_1 signs with a small amount of metadata, such as signing time, or if P_1 modifies N slightly to incorporate versioning information.) If our network primitives allow it, we can even request P_1's version of N rather than P_2's – effectively augmenting our "name" with structurally-identifiable information about the publisher.

The research question then becomes not what two legs of Zooko's Triangle do we choose, but given this general approach to securing even human-memorable names, how do we optimize our ability to locate and retrieve content based on those names – unique or not.

3 Details; *c.f.* Devil is in the

The advantage, for the postulator, of postulating a network security mechanism based on self-certifying names is that the hard part – getting the *correct* self-certifying name in the hands of the user in the first place is deemed out of scope – someone else's (as yet unsolved) problem.

Relying on an approach based on *certified content* – authenticated, signed, name-content mappings – entails navigating a different set of uncharted waters, itself historically labeled "Here There Be Dragons". In this case, the problem is not that of getting the user the correct name for their content – those names can be human-readable structured, and to a first approximation can be obtained through the same combination of memory, guessing, and lookup/search services we use to navigate the Web today. The problem is not retrieving the corresponding content based on the name – we've already delegated that to any number of potential network infrastructures. The problem is, how do we go from a name, and the resulting retrieved content to the user-focused properties of validity, provenance, and relevance we described in section 2. In other words, how does a user decide what content to trust?

We want to emphasize it is authenticity and trust which are key to effective content-based security, not secrecy. Given mechanisms to retrieve valid content by name, one can trivially add secrecy by encrypting that content using keys distributed and authenticated using those underlying protocols for retrieving valid content.

Helpfully, content-based security itself offers us tools to solve this problem. We can break the problem of trusting content down into several steps: first, verifying that a given name-content mapping was signed by a particular key; second, determining something about who that key belongs to – who, in user terms, published that content; and third, deciding whether or not that is an acceptable source for this particular content and the use to which it is to be put. In the terminology of section 2, the first step determines validity, the latter two steps determine provenance, and the name itself, along with the means by which it was obtained and the third step above, determine relevance.

3.1 The Mechanics of Publishing and Verifying Content

To make our approach concrete, we start by discussing the low-level mechanics. A content publisher must first determine the name by which she wants to refer to that content, which determines how that content will be found. Then she must generate a digital signature over that name and the corresponding content – $M_{(N,P,C)} = (N, C, Sign_P(N, C))$.[5] A content consumer, given N, must be able to retrieve not only C, but also the *authenticator* $Sign_P(N, C)$, and sufficient supporting information (signed or not) to determine what public key to use to verify $Sign_P(N, C)$, and where to find a copy of it if they don't have it already. (Similar problems are faced with key-verified data, but usually not mentioned.)

There are basically two ways to find the necessary key, illustrated by current standards for representing signed data: either you store with the data a copy of the key used to sign it, usually wrapped in a digital certificate to give the key itself some provenance (*e.g.* PKCS#7 (RFC 2315)), or you provide a name by which that key can be retrieved (*e.g.* XML Digital Signature Standard, which provides URLs to indicate where keys and certificates are stored). If the signature verifies, we have found the correct public key. If it does not, we know that we have incorrect or modified content, or we have the wrong key. Public

[4] As the signature is over the name, N, and the content, C, by the publisher P, changing any of N, C or P will result in a different signature.

[5] For simplicity of presentation, we describe content authentication in terms of a digital signature per content item. In practice, for efficiency we may individually sign fragments of content that will be separately retrieved to minimize verification latency, or we may aggregate many name-content mappings together using techniques such as Merkle Hash Trees [25] and sign them all together to minimize signing cost. Similarly, we gloss over here the question of how to seamlessly integrate content signing into the content production workflow.

keys act in this sense as a limited form of self-verifying content. (Note that this form of verification merely tells us that a given key signed this content mapping – not who that key belongs to or what value we should place on that signature.)

For example, a simple way to implement such a content-signing scheme in today's Web would be to sign web pages using a slight modification to standard XML digital signature mechanisms, and store and distribute links N to those signed content items – which contain C (the original HTML), $Sign_P(N, C)$, and a name (URL) at which one can find P's public key. The modification we require to achieve our secure *linkage* is to sign not just C, but N as well. This can actually be achieved quite simply, for example by designating an XML tag that indicates the name, N, of the content, and including that tag and N in the signed HTML (or XHTML). A verifier would then not only verify the signature, but also confirm that the name N that was signed matched the name N by which they found the content.

3.2 Determining Trust

The verification we described above is purely *syntactic*; it verifies that the indicated key was used to sign a particular mapping, but not what that key *means* – who it belongs to, or whether a signature by that key indicates whether or not the user should trust it. Luckily, the basic primitive of content-based security – authenticated mappings from names to content – can be used to implement mechanisms for establishing higher-level trust. This can be easily seen if one remembers that what we are effectively doing is *certifying* mappings from names to content. If the name N identifies not only the location of, but the owner of a public key, and the content C contains that public key, then the signed mapping $M_{(N,P,C)}$ is effectively an identity certificate where P acts as the certification authority.

The most common approach to determining trust on the Internet today is the one used by TLS to secure our bank example in section 1, namely a very simple, global Public Key Infrastructure (PKI). A small number of commercial providers whose keys are built into common web browsers serve as the roots of trust, signing digital certificates that associate web server DNS names with public keys. The good point of this system is that in a way it accidentally implies an association between trust and content. A traditional PKI serves to authenticate mappings between a name (*e.g.* of a person) and a public key, but says very little about what that key is or isn't allowed to do; that is left up to the consumer ("relying party" in PKI parlance). The PKI used on the Web associates a public key with a *content namespace* – the DNS domain name that begins the URLs that web server is trusted to serve. In other words, $www.amazon.com$ is trusted to serve content whose name begins with $www.amazon.com$ but not $www.paypal.com$. However, that PKI is not very expressive – *all* trust comes from outside the namespace, in the form of the commercial CAs. $www.amazon.com$ is not, for example, able to delegate control of parts of its namespace to other public keys.

One can trivially represent the current Web PKI in our model of signed name-content mappings. However, one can go beyond that, and achieve much greater representational power. By interpreting names N as *structured*, forming a forest of hierarchical namespaces (as is true of existing URLs), our content-based security scheme allows trust to be tightly coupled to content. For example, $nytimes.com$ could associate keys owned by department editors with different parts of its content namespace, allowing the entertainment editor to sign stories located under $/nytimes/entertainment$, but not $/nytimes/headlines$. Essentially we are representing trust in terms of a set of linked local namespaces, binding keys to related names. This mirrors well-studied paradigms for establishing user-friendly, namespace-based trust embodied in SDSI/SPKI [29, 18, 1, 12].

3.2.1 Evidence-Based Security

We can add to our notion of structured names a representation of *secure reference*, much like a trusted hyperlink or bookmark. Such a reference maps from a name N not to an arbitrary piece of content C, but to another name N', the *target* of that link, together with authentication information (*e.g.* the public key of the intended publisher P', or even the self-certifying digest of the intended content). Such references can be used to express *delegation*, saying in essence that the publisher P intends by the name N whatever the publisher P' refers to with the name N'.

Such references can express traditional forms of delegation, but they can also be used to build up a network of trust in content – if many publishers that we trust all say that they believe in P''s value for N', we are much more likely to believe it as well. If an attacker subverts a single publisher, *e.g.* obtaining the key for P'' and using it to forge a malicious value for N' the attack will fail, as the preponderance of the evidence will still point to the correct value. With each piece of additional signed support for trustworthy content, it becomes harder and harder for an attack to succeed, as an attacker simply cannot subvert all of the available evidence.

3.2.2 Bootstrapping Trust

Content-based security provides a number of powerful building blocks from which we can leverage trust in a small number of keys into trust in a large forest of interconnected content. But we must still start from some-

where – an initial set of keys, perhaps obtained via some out of band mechanism, from which we begin to build.

We would like to argue that the flexibility and expressive power of content-based security makes it easier to solve this problem too. We have shown above that one can easily replicate today's limited Web trust model in a content-based system; simply distribute the same root CA keys in the same out of band (browser installation) channel used today, and use those CA keys to certify root keys for each content namespace (as they certify web server keys today). However, associating keys with content allows us to differentiate among them, requiring higher assurance and greater trust for "important" content than unimportant, and allowing us to use different mechanisms to establish trust in different types of keys. I might trust my mother's public key because I obtained it from her by email and see her use it repeatedly over time; I might trust my notion of Google's public key because all of the people I know and web sites I visit believe in the same value of Google's key that I do.

Because we can refer to keys, policy and content securely by name, we can work to build a public system of trust that our applications can rely on [27].

4 Conclusions

We have suggested an improved approach to content naming, centered on authenticating the *linkage* between names and content rather than either names or content alone. This allows us to authenticate retrieved content regardles of how, or from whom, it is obtained, for any arbitrary (even human-readable) name form. We argue that this approach to certifying content is not only general, capable of securing content on both current connection-oriented and future content-centric networks, but deployable, and has the potential to enable a range of new applications.

References

[1] Martin Abadi. On SDSI's Linked Local Name Spaces. *Journal of Computer Security*, 6(1-2):3–21, October 1998.

[2] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. The Design and Implementation of an Intentional Naming System. *SIGOPS Oper. Syst. Rev.*, 33(5):186–201, 1999.

[3] Bengt Ahlgren, Matteo D'Ambrosio, Marco Marchisio, Ian Marsh, Christian Dannewitz, Börje Ohlman, Kostas Pentikousis, Ove Strandberg, René Rembarz, and Vinicio Vercellone. Design considerations for a network of information. In *ReArch*, 2008.

[4] Mark Allman. Personal Namespaces. In *HotNets-VI*, Atlanta, Georgia, November 2007.

[5] H. Balakrishnan, S. Shenker, and M. Walfish. Semantic-Free Referencing in Linked Distributed Systems. In *IPTPS*, February 2003.

[6] Hari Balakrishnan, Karthik Lakshminarayanan, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Michael Walfish. A Layered Naming Architecture for the Internet. In *SIGCOMM*, 2004.

[7] Matthew Caesar, Tyson Condie, Jayanthkumar Kannan, Karthik Lakshminarayanan, Ion Stoica, and Scott Shenker. ROFL: Routing on Flat Labels. In *SIGCOMM*, 2006.

[8] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Lecture Notes in Computer Science*, 2009:46, 2001.

[9] Christian Dannewitz, Kostas Pentikousis, René Rembarz, E. Renault, Ove Strandberg, and J. Ubillos. Scenarios and research issues for a network of information. In *Proc. 4th Int. Mobile Multimedia Communications Conf.*, July 2008.

[10] T. Dierks and C. Allen. *The TLS Protocol Version 1.0*. IETF - Network Working Group, The Internet Society, January 1999. RFC 2246.

[11] D. Eastlake. *Domain Name System Security Extensions*. IETF - Network Working Group, The Internet Society, March 1999. RFC 2535.

[12] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylonen. *SPKI Certificate Theory*, September 1999. RFC2693.

[13] Christian Esteve, Fábio L. Verdi, and Maurício F. Magalhães. Towards a new generation of information-oriented internetworking architectures. In *CONEXT*, 2008.

[14] Bryan Ford, Jacob Strauss, Chris Lesniewski-Laas, Sean Rhea, Frans Kaashoek, and Robert Morris. Persistent Personal Names for Globally Connected Mobile Devices. In *OSDI*, 2006.

[15] Bryan Ford, Jacob Strauss, Chris Lesniewski-Laas, Sean Rhea, Frans Kaashoek, and Robert Morris. User-relative names for globally connected personal devices. In *IPTPS*, 2006.

[16] Kevin Fu, M. Frans Kaashoek, and David Mazières. Fast and secure distributed read-only file system. *ACM Trans. Comput. Syst.*, 20(1):1–24, 2002.

[17] Mark Gritter and David R. Cheriton. An architecture for content routing support in the internet. In *Usenix Symposium on Internet Technologies and Systems (USITS)*, 2001.

[18] Joseph Y. Halpern and Ron van der Meyden. A logic for SDSI's linked local name spaces. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, 1999.

[19] Van Jacobson. A New Way to Look at Networking, August 2006. `http://video.google.com/videoplay?docid=-6972678839686672840&ei=iUx3SajYAZPiqQLwjIS7BQ&q=tech+talks+van+jacobson`.

[20] Van Jacobson. Making the Case for Content-Centric Networking: An Interview with Van Jacobson. *ACM Queue*, January 2009.

[21] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A Data-Oriented (and Beyond) Network Architecture. In *SIGCOMM*, 2007.

[22] John Kubiatowicz et al. OceanStore: An architecture for global-scale persistent storage. *SIGPLAN Not.*, 35(11):190–201, 2000.

[23] Laurent Mathy and Luigi Iannone. LISP-DHT: towards a dht to map identifiers onto locators. In *CONEXT*, 2008.

[24] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating Key Management from File System Security. In *SOSP*, 1999.

[25] Ralph Charles Merkle. *Secrecy, authentication, and public key systems.* PhD thesis, 1979.

[26] Börje Ohlman et al. First NetInf architecture description, April 2009. `http://www.4ward-project.eu/index.php?s=file_download&id=39`.

[27] Eric Osterweil, Dan Massey, Batsukh Tsendjav, Beichuan Zhang, and Lixia Zhang. Security Through Publicity. In *HOTSEC '06*, 2006.

[28] Bogdan C. Popescu, Maarten van Steen, Bruno Crispo, Andrew S. Tanenbaum, Jan Sacha, and Ihor Kuz. Securely replicated web documents. In *IPDPS*, 2005.

[29] Ronald L. Rivest and Butler Lampson. SDSI - A Simple Distributed Security Infrastructure. Technical report, MIT, 1996.

[30] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *SIGCOMM*, 2001.

[31] D. Trossen. Conceptual architecture of PSIRP including subcomponent descriptions (D2.2), June 2008. `http://psirp.org/publications`.

[32] Michael Walfish, Hari Balakrishnan, and Scott Shenker. Untangling the Web from DNS. In *NSDI*, 2004.

[33] Zooko Wilcox-O'Hearn. Names: Decentralized, secure, human-meaningful: Choose two, September 2003. `http://zooko.com/distnames.html`.

[34] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.