

# Commercial authentication

By Luis von Ahn, Manuel Blum, and John Langford

TELLING  
HUMANS AND  
COMPUTERS  
APART  
AUTOMATICALLY

a CAPTCHA is something more than just an image with distorted text: it is a test, any test, that can be automatically generated, which most humans can pass, but that current computer programs cannot pass. Notice the paradox: a CAPTCHA is a program that can generate and grade tests that it itself cannot pass (much like some professors).

- ① Computer generated
- ② People have to be able to succeed
- ③ Computers MUST fail

CAPTCHA stands for “Completely Automated Public Turing Test to Tell Computers and Humans Apart.” The P for Public means that the code and the data used by a CAPTCHA should be publicly available. This is not an open source requirement, but a security guarantee: it should be difficult for someone to write a computer program that can pass the tests generated by a CAPTCHA even if they know exactly how the CAPTCHA works (the only hidden information is a small amount of randomness utilized to generate the tests). The T for “Turing Test to Tell” is because CAPTCHAs are like Turing Tests [10]. In the original Turing Test, a

human judge was allowed to ask a series of questions to two players, one of which was a computer and the other a human. Both players pretended to be the human, and the judge had to distinguish between them. CAPTCHAs are similar to the Turing Test in that they distinguish humans from computers, but they differ in that the judge is now a computer. A CAPTCHA is an *Automated* Turing Test.

Modern cryptography has shown that open or intractable problems in number theory can be useful: an adversary cannot act maliciously unless he can solve an open problem (like factor a very large number). Similarly, CAPTCHAs show that open problems in AI can be useful: adversaries cannot vote thousands of times in online polls or obtain millions of free email accounts unless they can solve an open problem in AI.

In the case of ordinary cryptography, it is assumed (for example) that the adversary cannot factor 1024-bit integers in any reasonable amount of time. In our case, we assume the adversary cannot solve an artificial intelligence problem with higher accuracy than what's currently known to the AI community [1, 2, 5, 6, 8]. This approach has the beneficial side effect of inducing security researchers, as well as otherwise malicious programmers, to advance the field of AI (much like computational number theory has been advanced since the advent of modern cryptography). This is how lazy cryptographers do AI.

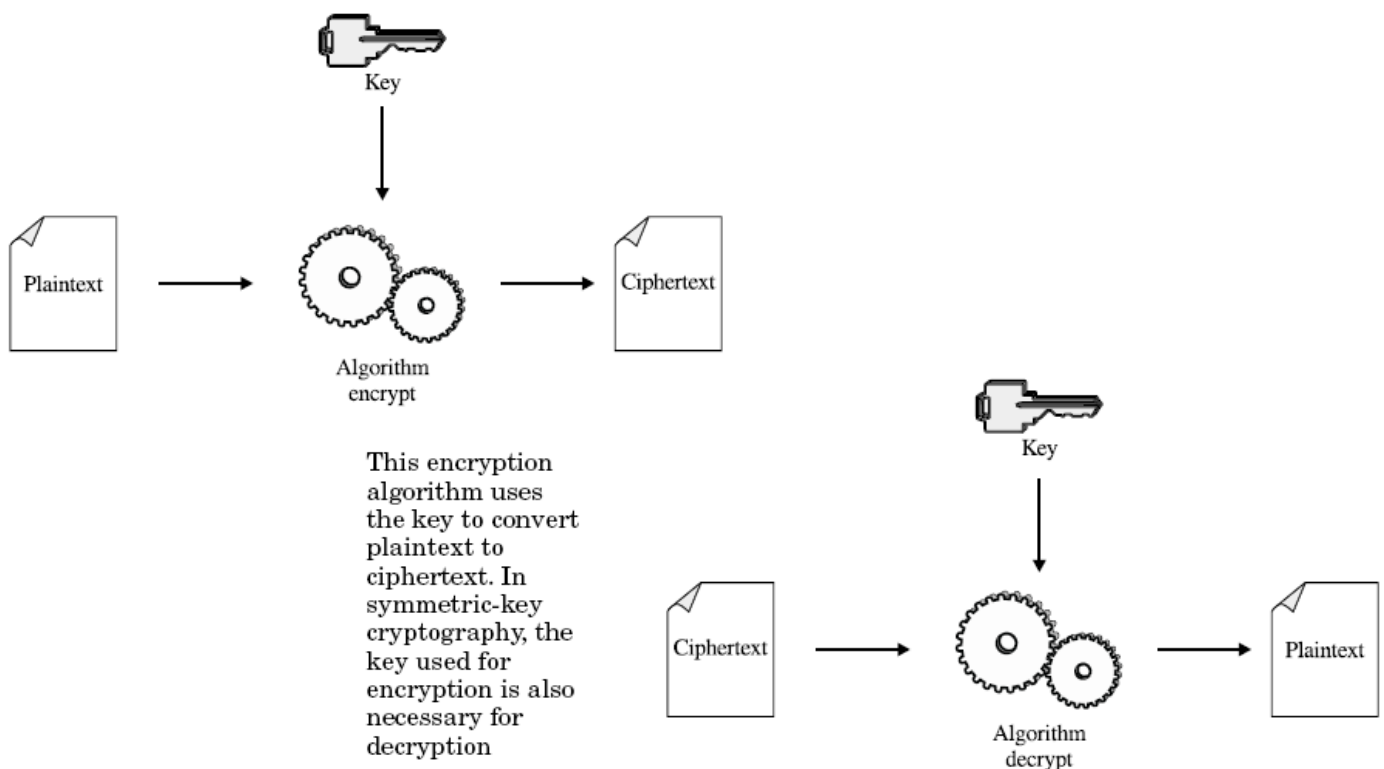
# RSA Security's Official Guide to CRYPTOGRAPHY

Learn how secure data-encryption techniques work

Protect confidential information on your network

Get official current cryptography standards on enclosed CD-ROM

**Steve Burnett & Stephen Paine**



A Worse Than Worst-Case Scenario: How Long a Brute-Force Attack Will Take for Various Key Sizes

Bits	1 percent of Key Space	50 percent of Key Space
56	1 second	1 minute
57	2 seconds	2 minutes
58	4 seconds	4 minutes
64	4.2 minutes	4.2 hours
72	17.9 hours	44.8 days
80	190.9 days	31.4 years
90	535 years	321 centuries
108	140,000 millennia	8 million millennia
128	146 billion millennia	8 trillion millennia

## Why Is a Key Necessary?

All computer crypto operates with keys. Why is a key necessary? Why not create an algorithm that doesn't need a key?

As you saw in the memo example, if attackers can understand the algorithm, they can recover secret data simply by executing the algorithm. That's like installing a deadbolt on your front door with the lock on the outside. It's true that when the deadbolt is in place, the door cannot be opened. But anyone can open the door simply by turning the lock.

It might seem that the solution is to keep the algorithm secret, but that approach has several problems. First, attackers always crack the algorithm (see "Historical Note: They Always Figure Out The Algorithm," later in this chapter). What's more, suppose you do manage to keep the algorithm secret. Unless you are a cryptography expert and develop your own algorithms, you also must trust the company that wrote your algorithm never to reveal it deliberately or accidentally. Does anyone have that much trust in a corporate entity?

Here's the real question: Which would you trust more to keep secrets—an algorithm that must be kept secret, or an algorithm that can do its job even if everyone in the world knows exactly how it works? That's where keys come in.

Keys relieve you of the need to worry about the algorithm used in your encryption scheme. If you protect your data with a key, you need protect only the key, something that's easier to do than protecting an algorithm. In this book you'll learn a lot about key protection. Also, if you use keys to protect your secrets, you can use different keys to protect different secrets. This means that if someone breaks one of your keys, your other secrets are still safe. If you're depending on a secret algorithm, an attacker who breaks that one secret gets access to all your secrets.

Fred B. Schneider

*Samuel B. Eckert Professor of Computer Science*

Department of Computer Science  
4115C Upson Hall, Cornell University  
Ithaca, NY 14853

## CS5430: System Security

### Authentication using Shared Key Cryptography

An *authentication protocol* allows a principal receiving a message to determine which principal sent that message. Any aficionado of spy movies is doubtless familiar with one means by which principals might authenticate each other: The first principal asks an innocuous question --- called the *challenge* --- of the second ("Its a bright sunny day in Dover."). And if the first principal knows the secret response to that challenge ("But the queen prefers rain.") then the first concludes that the second is a compatriot.

This authentication protocol has a significant weakness. Anyone overhearing the exchange is thereafter able to impersonate either side. The basic idea---using knowledge of a secret---is sound. The problem is that a principal must reveal the secret in order to prove knowledge of that secret. This is known as *weak authentication*, and it is subject to *replay attacks* whereby an adversary repeats fragments of a past protocol run, appearing to have knowledge of the secret, and thus passes the authentication test.

In a *strong authentication* protocol, knowledge of the secret is demonstrated without revealing the secret itself. If, for example, the secret is used in determining the answer for the challenge, then an adversary overhearing the challenge and response is not going to be able to feign knowledge of the secret when confronted with other, different challenges.

One way to implement strong authentication in a network starts from the assumption that principals share a secret key. Knowledge of that key is demonstrated by using the key to encrypt and decrypt challenges, but the key itself is never revealed. Here is a protocol that allows  $B$  to authenticate  $A$ . Assume that  $A$  and  $B$  are the only two principals with knowledge of secret key  $k$ .

1.  $B$ : select and store a new random value  $r$ .
2.  $B \rightarrow A$ :  $B, r$
3.  $A \rightarrow B$ :  $\{r\}_k$
4.  $B$ : check whether  $D(k, \{r\}_k)$  equals the stored value  $r$  from step 1.

Random value  $r$  selected in step 1 is called a *nonce*. It is (by design) new, so an attacker who has recorded previous versions of message 3 is unable to replay one of those messages in order to satisfy this new challenge issued by  $B$ . Inclusion of " $B$ " in step 2 allows receiver  $A$  to select the correct shared key (i.e., the one shared with  $B$ ) for generating message 3. An attacker seeing message 2 bearing the challenge does not know secret key  $k$  and thus should not be able to generate a response that satisfies  $B$  (in step 4) for the challenge (issued in step 2).

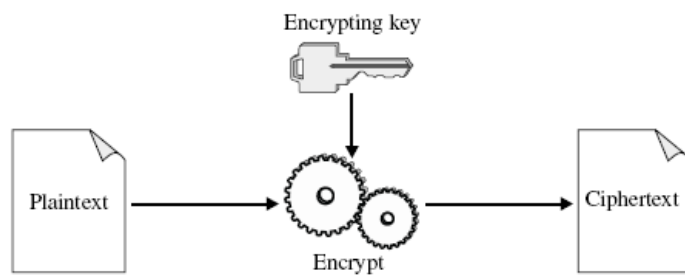
# RSA Security's Official Guide to CRYPTOGRAPHY

Learn how secure data-encryption techniques work

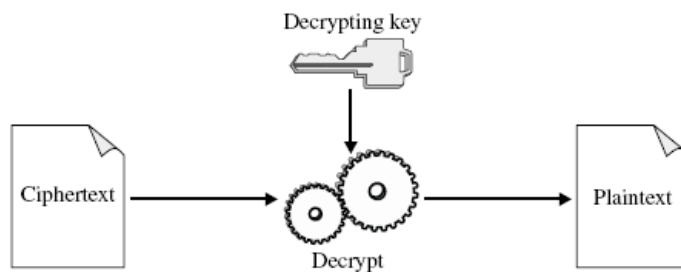
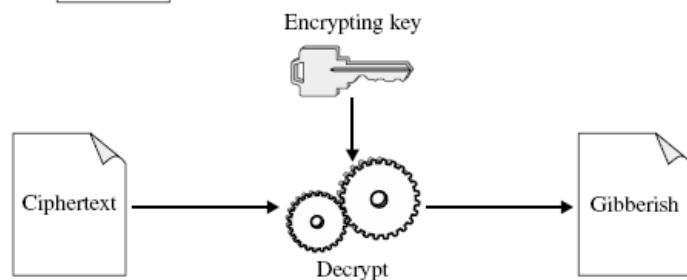
Protect confidential information on your network

Get official current cryptography standards on enclosed CD-ROM

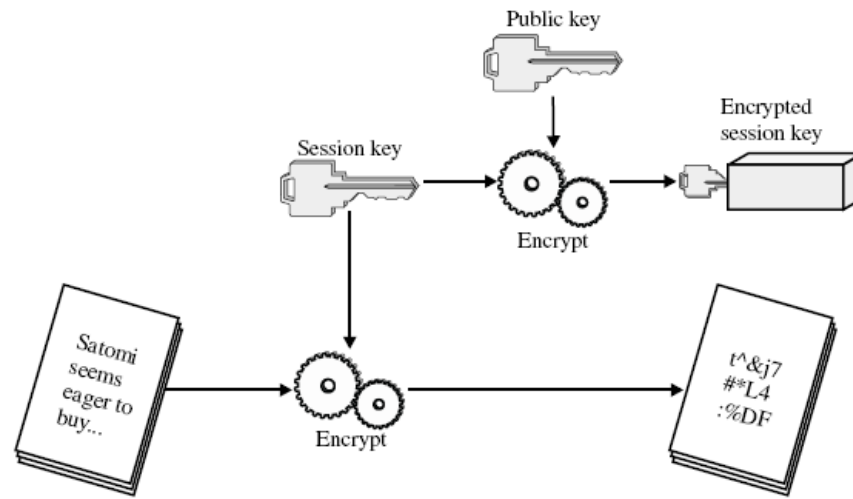
**Steve Burnett & Stephen Paine**



In asymmetric crypto, the encrypting key cannot be used to decrypt; you must use its partner

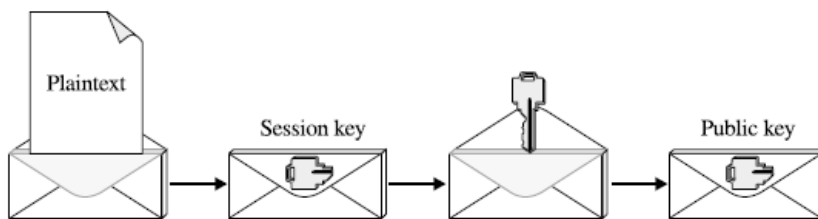


You use a session key with a symmetric algorithm to encrypt the bulk data and then encrypt the session key with the recipient's public key



This process of encrypting bulk data using symmetric-key crypto, and encrypting the symmetric key with a public-key algorithm, is called a *digital envelope*. The idea is that the symmetric key is wrapping the data in an envelope of encryption, and the public key is wrapping the symmetric key in an envelope (see Figure 4-6).

A digital envelope. The session key wraps the bulk data in an envelope of encryption, and the public key wraps the session key in another envelope



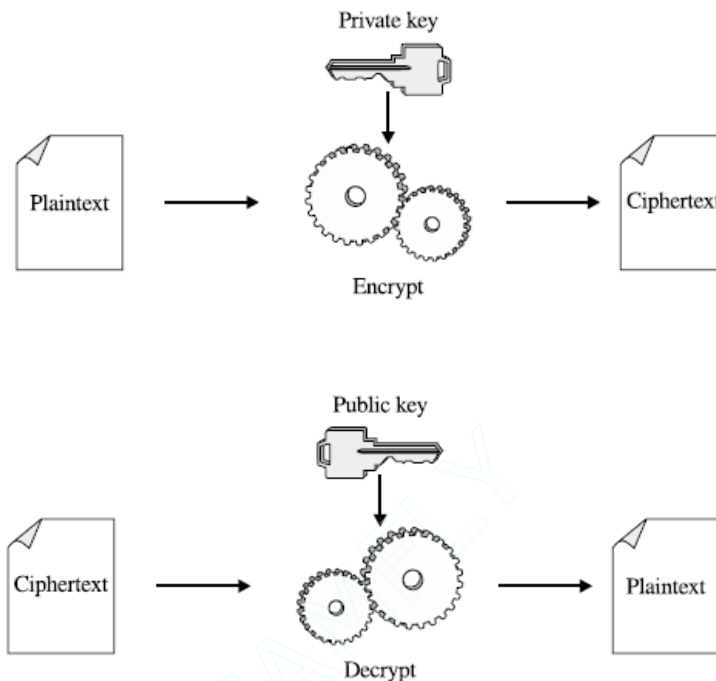
Time to Break Keys of Various Sizes with \$10 Million to Spend

Symmetric Key (Size in Bits)	ECC Key (Size in Bits)	RSA Key (Size in Bits)	Time to Break	Number of Machines	Amount of Memory
56	112	430	Less than 5 minutes	105	Trivial
80	160	760	600 months	4,300	4GB
96	192	1,020	3 million years	114	170GB
128	256	1,620	10 <sup>16</sup> years	0.16	120TB

The table says that with \$10 million, an attacker could buy 105 specially made computers to crack a 56-bit symmetric key, a 112-bit ECC key, or a 430-bit RSA key in a few minutes. Actually, that \$10 million would probably buy more than 105 machines, but 105 is all it would take. With the same amount of money, at the next key level the attacker could buy 4,300 machines specially built to solve the problem; at the next key level, 114, and at the next level, 0.16.

Why does the money buy fewer machines as the key size increases? The reason is that the amount of required memory increases. The base computer is the same, but to break bigger keys, the attacker needs more memory (120 terabytes, or about 120 trillion bytes, in the case of a 1,620-bit RSA key), and buying memory would eat up the budget. In fact, the attacker will probably need more than \$10 million to break a 1,620-bit RSA key because that amount of money would only buy 0.16, or about 1/6, of a machine.

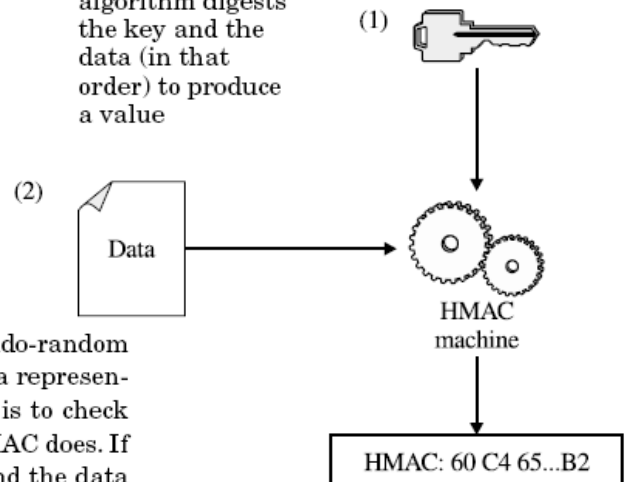
If you encrypt plaintext with an RSA private key, you can use the public key to decrypt it



When you use the RSA algorithm, it means that anything encrypted with the public key can be decrypted only with the private key. What would happen if you encrypted plaintext with a private key? Is that possible? And if so, which key would you use to decrypt? It turns out that RSA works from private to public as well as public to private. So you can encrypt data using the private key, and in that case, only the public key can be used to decrypt the data (see Figure 5-1).

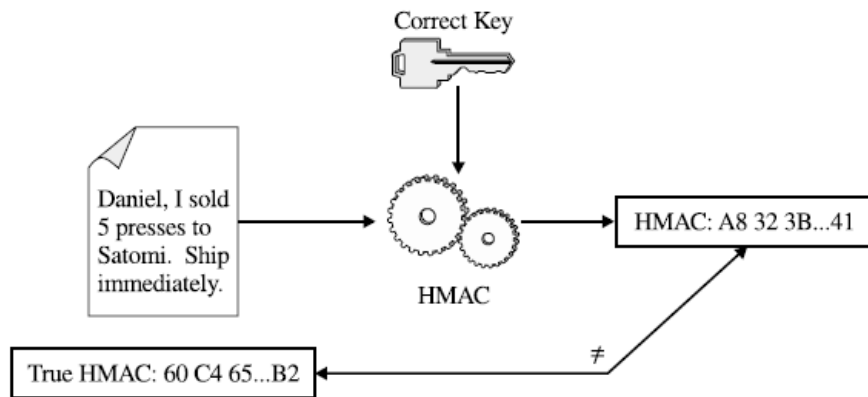
You may ask, “What good is that?” After all, if you encrypt data with your private key, anyone can read it because your public key, which is publicly available, can be used to decrypt it. It’s true that using RSA in this direction does not let you keep secrets, but it is a way to vouch for the contents of a message. If a public key properly decrypts data, then it must have been encrypted with the private key. In the crypto community, this technique is conventionally called a *digital signature*.

The HMAC algorithm digests the key and the data (in that order) to produce a value

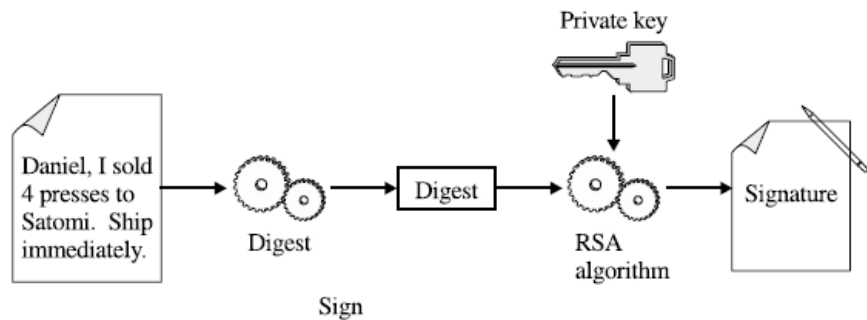
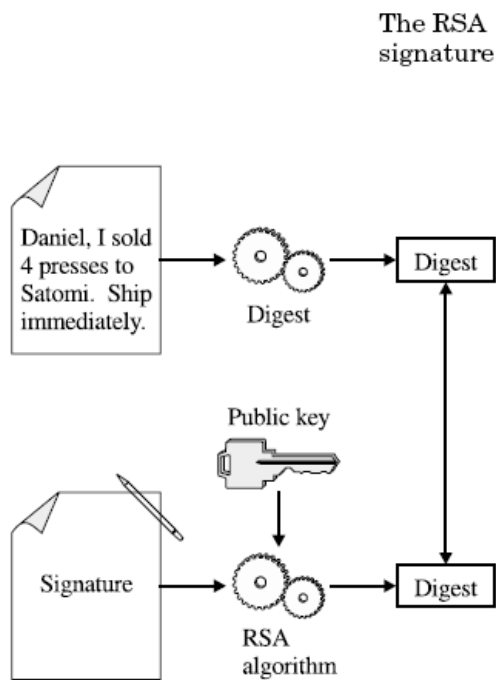


We've described a message digest as the foundation of a pseudo-random number generator or password-based encryption, and now as a representative of a larger message. Another use for a message digest is to check *data integrity*, which is the term used to describe what the HMAC does. If you're concerned that the information may be altered, you send the data along with a check. If the message was altered, the check will also be different. Of course, you must ensure that the check value cannot be altered to match any changes in the message.

If the check value shows no alterations, the data has been shown to have integrity. "Integrity" is a word for honest, sound, and steadfast. When used in relationship to data, it may seem pretentious, but it does describe data that you can count on, at least in terms of its authenticity.



Daniel digests the correct key but the wrong message, so he knows that something is wrong

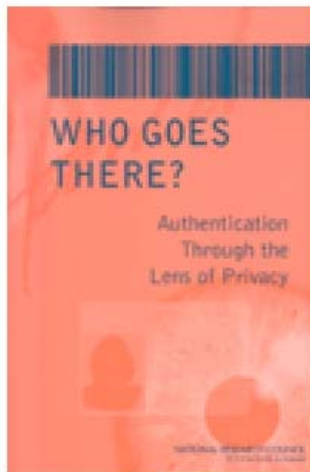


Are they the same?

It works like this. Pao-Chi digests the message and then encrypts the digest with his private key. He sends Daniel the message along with the encrypted digest, which serves as the signature. Daniel separates the two components and digests the message he received. He has a message in his possession and knows the digest that will produce it (he just computed it). He must determine whether the message he now has is the same message Pao-Chi sent. If Daniel knew what Pao-Chi computed as a digest, he could make that determination. Well, he has Pao-Chi's digest—it's the signature. So Daniel uses Pao-Chi's public key to decrypt the signature. That's the value Pao-Chi signed (see Figure 5-10). Is it the same answer Daniel got? If it is, he knows that the data was not altered in transit and that Pao-Chi is vouching for the contents.

Notice something powerful about the digital signature: Each chunk of data has its own signature. This means that no single digital signature is associated with an individual or key pair. Each signature is unique to the data signed and the keys used. When an individual signs two messages with the same key, the signatures will be different. Moreover, when two people with different keys sign the same data, they will produce different signatures. As a result, someone cannot take a valid signature and append it to the bottom of a different message, something that makes it much more difficult to forge a signature.

Think of it this way. Two people (a sender and a receiver) each have a copy of a message. Are they really copies or was the receiver's copy altered in transit? To find out, they digest the two messages and compare them. If the digests are the same, both parties know that the two versions match. If the digests don't match, something went wrong. How do you know that the sender's digest was not altered? You know that because it was encrypted with the sender's private key. How do you know that it was encrypted with the sender's private key? You know it because the public key decrypts it.



## Who Goes There?: Authentication Through the Lens of Privacy

Stephen T. Kent and Lynette I. Millett, Editors,  
Committee on Authentication Technologies and Their  
Privacy Implications, National Research Council

ISBN: 0-309-52654-X, 232 pages, 8 1/2 x 11, (2003)

**This PDF is available from the National Academies Press at:**  
<http://www.nap.edu/catalog/10656.html>

A crucial issue for authentication technologies is whether they are inherently centralized or decentralized. This distinction affects both their deployability and their privacy implications.

Some technologies require little or no infrastructure; any system can make use of such technologies without relying on additional systems for support. This is one of the major drivers of the use of static passwords: They are extremely easy to set up in a highly localized fashion. An application can create and maintain its own password database with about the same effort as that needed for maintaining a database of authorized users. (In fact, doing so properly is rather more complex, but there are numerous poorly implemented password systems.) Some public key authentication technologies have similar decentralized properties. Some challenge/response protocols are designed for local use and require minimal infrastructure. The one-time password<sup>18</sup> and Secure Shell (SSH)<sup>19</sup> protocols are good examples. The latter makes use of public key cryptography but not a public key infrastructure (described later in this section). Both arguably provide a more secure authentication capability than passwords, but they are still intended for use in local contexts, such as a single computer or at most a single organization.

Some types of authentication technologies require some degree of centralization—for example, to help amortize the costs associated with deployment to gain security benefits. Kerberos and public key infrastructure (PKI) are good examples of such systems. In Kerberos, the key distribution center (KDC) is a centralized infrastructure component that stores the passwords of all users, preventing them from having to be shared with each system to which a user connects. The KDC is aware of all sites with which the user interacts, because the KDC is invoked the first time that a user establishes a connection in any given log-in session. The content sent over the connections is not necessarily revealed; nevertheless, the central site operates as the verifier, acting as an intermediary between the user (presenter) and the sites that rely on Kerberos for authentication. The scope of a Kerberos system deployment is typically limited, which in practice mitigates some of the privacy concerns. Although Kerberos systems can be interconnected, most usage of Kerberos is within an individual organization. When cross-realm authentication is employed, only those transactions that involve multiple realms are known outside a user's home realm.<sup>20</sup> This limits the adverse privacy aspects of using such a system. However, if a single Kerberos realm was used to authenticate individuals to systems across organizational boundaries, the privacy implications would be much worse. Thus, the same technology can be used in different contexts with vastly different privacy implications. For more information about Kerberos, see Figure 5.1.

The Passport and Liberty systems, though very different in detail, are centralized systems designed expressly to authenticate large user populations to a wide range of disparate systems, with attendant privacy impli-

The Passport and Liberty systems, though very different in detail, are centralized systems designed expressly to authenticate large user populations to a wide range of disparate systems, with attendant privacy implications. While their designs differ slightly, both offer users the same basic feature: the convenience of single sign-on to a variety of Web services. From a privacy perspective, the obvious drawback to centralized authentication systems is that all Web clients cannot be expected to trust the same authentication service with what could be personally identifying information. Passport and Liberty both address this fundamental obstacle by allowing what they call a federated topology. "Federated," in this context, means that peer authentication services can interoperate with different subsets of service providers. For example, a car rental company could rely on four different airlines' authentication services. Theoretically, a single user could navigate seamlessly between multiply affiliated sites after authenticating only once. Even in their federated form, however, there are two types of privacy risk inherent in these single sign-on systems: exposure of what we call identity data (the set of all information associated with an individual within this identity system) by the authentication service and the aggregation of an entity's (or his or her identifier's) downstream behavior.

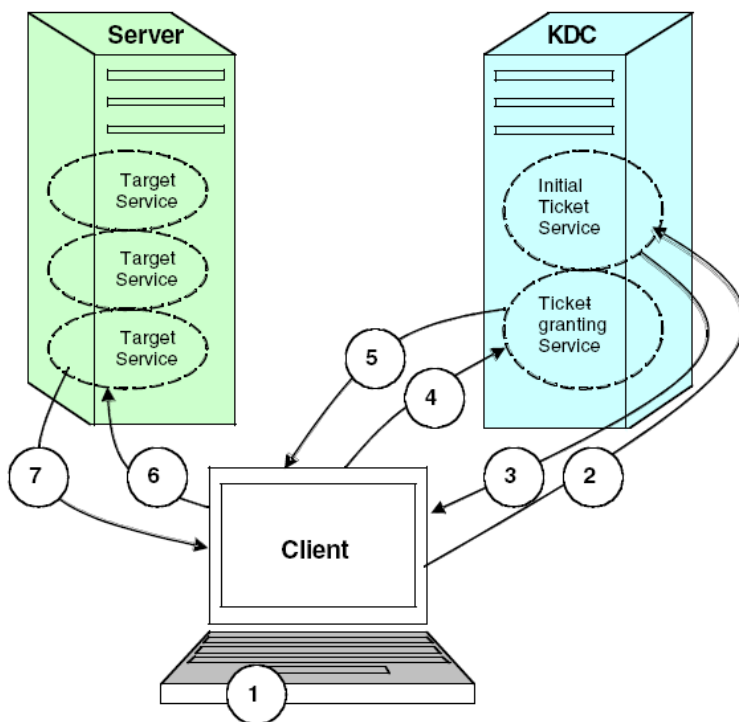


FIGURE 5.1 Kerberos:

1. User provides a principal (user name) and password to the client system.
2. Client queries the Initial Ticket Service of the Kerberos key distribution center (KDC) for a ticket-granting ticket (TGT), which will allow the client to request tickets for specific services later on. The client's request includes a derivative of the user's password, which the Initial Ticket Service verifies.
3. The KDC's Initial Ticket Service provides the client with a dual-encrypted initial TGT containing a log-in session key. The client system converts the user's password into an encryption key and attempts to decrypt the TGT.
4. The client uses the TGT and the log-in session key to request tickets to specific services from the KDC's Ticket-Granting Service.
5. The Ticket-Granting Service decrypts the TGT with its own key, and then decrypts the service request using the TGT's session key. If decryption is successful on both counts, the Ticket-Granting Service accepts the user's authentication and returns a service ticket and a service-session key (encrypted with the log-in session key) for the targeted service. This result can be cached and reused by the client.
6. The client uses the log-in session key provided in step 3 to decrypt the service ticket, gaining access to the service-session key. This key is then used to request access to the target service. This request is accompanied by an encrypted time stamp as an authenticator.
7. Access to the target service is granted. Steps 4 through 7 can be repeated when access to other services is needed; service messages can be encrypted with the service-session key. A time limit is built into the log-in session in steps 3 and 5; the user will need to enter the password again when the log-in session has timed out.

## authentication architectures

centralized — single issuer  
— single verifier

federated — ALWAYS  
multiple

de-centralized — SOMETIMES  
multiple issuer/verifier

security

cost

performance

scalability

population

management

maintenance

**Recommendation 5.3: Public key infrastructures should be limited in scope in order to simplify their deployment and to limit adverse privacy effects. Software such as browsers should provide better support for private (versus public) certificate authorities and for the use of private keys and certificates among multiple computers associated with the same user to facilitate the use of private certificate authorities.**

This analysis suggests that authentication technologies that imply some degree of centralization can be operated over a range of scales with vastly differing privacy implications. Thus, neither Kerberos nor PKI intrinsically undermines privacy (beyond the fact that they are authentication systems and as such can affect privacy), although each could be used in a way that would do so. In general, decentralized systems tend to be more preserving of privacy: No single party has access to more than its own transaction records. An individual may use the same password for two different Web sites; for a third party to verify this, the party would need at least the cooperation of both sites and (depending on the precise password storage technology being used) perhaps special-purpose monitoring software on both sites. But if users employ the same identifiers at each site, the potential for privacy violations is significantly increased. This same observation applies to any form of decentralized authentication system.

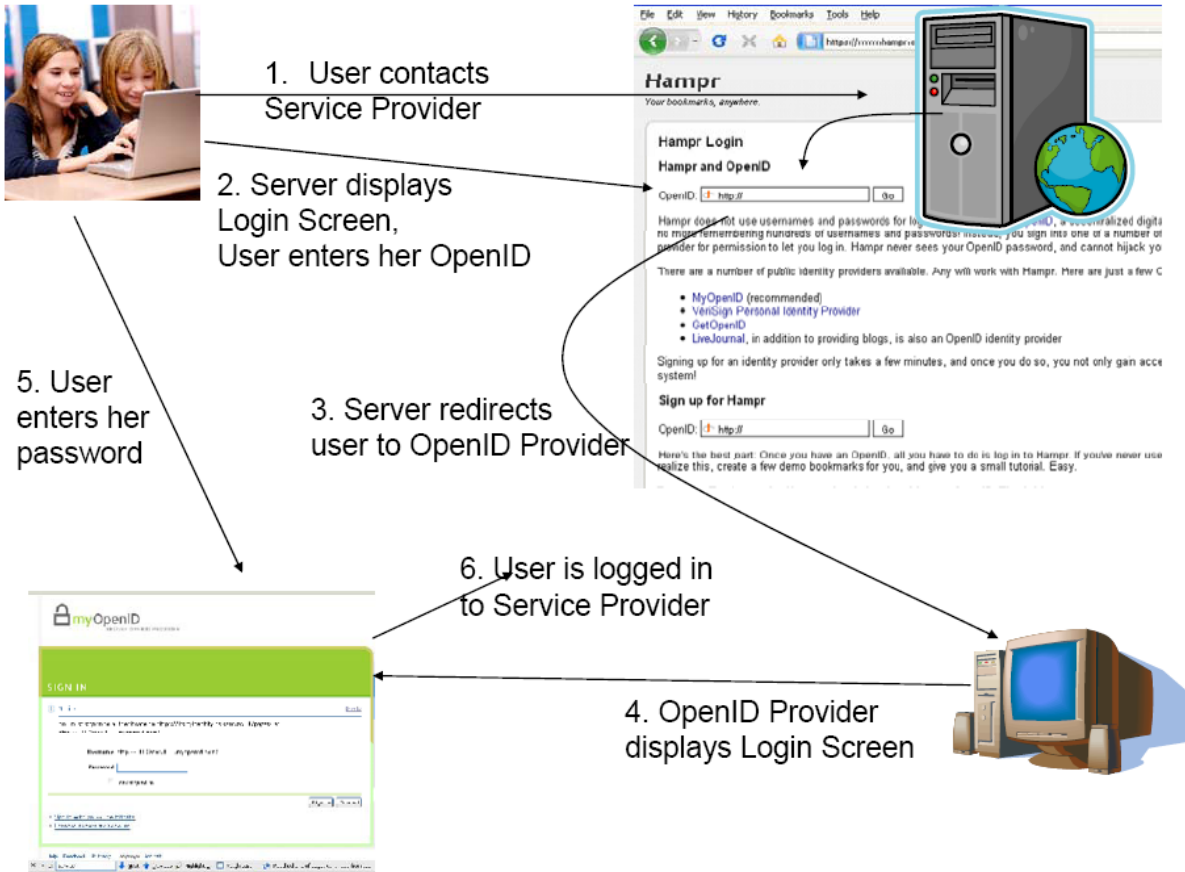
An essential requirement for preserving privacy in authentication systems is allowing an individual to employ a different identifier when he or she asserts a different identity—for example, in different organizational contexts. The use of different identifiers makes it harder to correlate the individual's activities across systems, which helps preserve privacy. This goal can be achieved with technologies ranging from passwords to PKIs. Also, if each system collects less personal information on its users—only what is required to satisfy the requirements of that system—this, too, is privacy-preserving.

**Finding 5.6: Core authentication technologies are generally more neutral with respect to privacy than is usually believed. How these technologies are designed, developed, and deployed in systems is what most critically determines their privacy implications.**



## JISC Final Report

Project Information			
<b>Project Acronym</b>	OpenID Study		
<b>Project Title</b>	Review of OpenID		
<b>Start Date</b>	3 December 2007	<b>End Date</b>	30 November 2008
<b>Lead Institution</b>	EDINA, University of Edinburgh		
<b>Project Director</b>	Peter Burnhill		
<b>Project Manager &amp; contact details</b>	Sandy Shaw Senior Technical Officer Email: s.shaw@ed.ac.uk Address: EDINA, University of Edinburgh Causewayside House EDINBURGH EH9 1PR Tel: 0131 650 4988 Fax: 0131 650 3308		
<b>Partner Institutions</b>	ISSRG, University of Kent		
<b>Project Web URL</b>	<a href="http://sdss.ac.uk/">http://sdss.ac.uk/</a>		
<b>Programme Name (and number)</b>	JISC eInfrastructure Programme		
<b>Programme Manager</b>	James Farnhill		



OpenID (see <http://www.openid.net/>) is an authentication system that is based on the premise that anyone can have a URL (or alternatively an Extensible Resource Identifier (XRI) [5] which is allowed in version 2.0) and an OpenID Identity Provider (OP) which is willing to speak on behalf of this URL or XRI. During its short lifetime, OpenID has evolved through three versions, namely Open ID v1.0, v1.1 [3] and v2.0 [2]. Whilst the first two versions were only concerned with authentication, v2.0 has added the capability for the exchange of identity attributes as well [4]. The first version of OpenID (v.1.0) had several security weaknesses some of which were quickly patched in v1.1 (e.g. messages could be replayed), while others were fixed in v2.0. However, as described below, v2.0 still suffers from several security weaknesses, which may or may not pose a significant risk, depending upon the application that one wishes to secure with OpenID.

In brief, OpenID works as follows (see Figure 1). When a user contacts a Service Provider (SP) that supports OpenID, he presents his URL (or XRI), and the SP contacts the URL to see who is the OP that speaks for it. This is the process of Identity Provider Discovery, and it bypasses the need for the Where Are You From service of Shibboleth and the Identity Selector in CardSpace. When XRIs are used, these similarly are resolved by the SP to find the OP that can authenticate the user.

Once the identity provider has been discovered, the SP must establish a shared secret with it so that future messages can be authenticated, using the well known process of message authentication codes (MAC). The OpenID specifications use Diffie-Hellman to establish the shared secret between the OP and SP; unfortunately, Diffie-Hellman is vulnerable to man in the middle attacks (see 5.1.2.9 below). Once the OP and SP have established a shared secret, the SP redirects the user to the OP, to be authenticated by any mechanism deemed appropriate by the OP. During the authentication process the OP is supposed to check that the user wants to be authenticated to this SP, by displaying the “realm” of the SP to the user. (The realm is a pattern that represents part of the name space of the SP e.g. \*.kent.ac.uk). But this realm information can easily be spoofed by an evil SP, which will lull the user into a false sense of security that they are authenticating to a known SP when in fact they will be redirected to an evil one after the authentication has completed

After successful user authentication, the OP redirects the user back to the SP along with an authentication token saying that the user has been authenticated and has control over the OpenID they specified. The SP then grants the user access to its services, as it deems appropriate.

## **5.1.2. Security Weaknesses in OpenID**

### **5.1.2.1. No Authentication of the User at Registration Time**

Firstly there is no requirement to validate the person or entity which is allocated an OpenID URL. An OpenID URL can belong to anyone (or any process). This is equivalent to a Level of Assurance [6] of zero. This is a spammers’ paradise, since spammers will be able to register thousands of OpenIDs with no restrictions whatsoever. The SP has no indication of who the user (or spammer) might be. Thus using OpenID authentication only, the SP is not able to authorise the user to access any resource of any value. Only publicly available resources may be accessed. The OpenID Attribute Exchange protocol [4] is an attempt to rectify this deficiency, by allowing the SP to request additional user attributes from the OP. However, the exchange of user attributes, that are to be used for authorisation, requires a trust infrastructure. SPs need to trust that OPs have strong registration procedures and have validated that the user is entitled to be assigned the particular set of attributes. Currently OpenID has no mechanism for discerning between OPs, as described below.

### 5.1.2.2. No Trust Infrastructure

OpenID has no mechanisms for specifying or enforcing trust infrastructures. Worse than that, because *“the OpenID technology is not proprietary and is completely free”* then users, OPs and SPs will expect that all OpenID providers are equal and can be used interchangeably. Once we start to introduce trust infrastructures, then by definition not everyone is equally trustworthy any more. Some OPs may strongly authenticate users at registration time, others may do it weakly, and others not at all, which means that some SPs will trust some OPs more than others, and may not trust some OPs at all. This could lead to massive confusion in the market place, since users will not know which OpenID and OP they should use for which SP, which will lead to them having to register with multiple OPs, which negates the purpose of having one *“OpenID [which] can stay with you, no matter which Provider you move to”*. Obviously in time, users will discern which OPs are more trustworthy than others, and which OpenID/OP combination gets them access to more services (it will typically be the most trustworthy OP). This could have the positive effect of the market place forcing OpenID providers to become more trustworthy and to have much stronger registration procedures for their users, but it will take time. Universities are clearly at an advantage in this respect compared to existing OpenID providers, since universities already have relatively strong registration procedures for their staff and students. But conversely, universities are at a disadvantage compared to banks and some government agencies, which have much stronger registration procedures. So there is clearly a continuum from zero trust to high trust when it comes to evaluating OPs and IdPs.

By way of comparison, the UK federation has an optional requirement that members may claim compliance with, which specifies that the real world identity of a user authenticated by the IdP can always be determined by the home institution. Trust relationships between SPs and IdPs are therefore built-in to the UK federation.

### 5.1.2.3. Re-allocation of Your OpenID

One of OpenID's key selling points initially was that the same OpenID URL would always be used by the same person or entity, so that an SP would know it was always talking to the same entity. Unfortunately this is no longer true. The OpenID URLs that are assigned to people by OpenID Providers are not guaranteed to be theirs forever. For example, if a user has the OpenID of *myidentifier.americaonline.com* which remains unused for a given period, America Online has stated publicly that there is no constraint on assigning the user's OpenID to another customer. This means that the binding of a URL to a person is not permanent, and that any SP or user that was expecting an OpenID URL to belong to just one user may find that, unbeknown to them, another user has now inherited the same URL. This could have serious consequences, for example, if a user had stored confidential or private information at a SP under his (now reallocated) OpenID. Until this issue is successfully resolved, the advice to users must be *“don't store valuable information under your OpenID, or if you do, make sure you remove the information before you give up your OpenID”*.

#### 5.1.2.4. No Privacy Protection

Because a user uses the same OpenID URL for every session, and every SP interacts with the user's OP for every session, the OP is able to fully track all the SPs that the user interacts with. This probably explains, at least partially, why every commercial web service provider, and large players such as Google, wants to be an OP. They want to grab the users for themselves, and then follow their web behaviours and interaction patterns. SPs are similarly able to track users between sessions, since the user will always be authenticated with the same OpenID. Perhaps the saving grace is that no-one, not even the OP, knows exactly who the user is (see 1 above.)

Shibboleth, as with any SSO system, suffers from some loss of privacy, in that the Shibboleth IdP is not only able to fully track all the SPs that the user interacts with, but will often know who the user is. Where Shibboleth is better, is that SPs are not able to correlate a user's activity across different services, since the user presents a different persistent identifier to each SP (by means of eduPersonTargetedID).

CardSpace on the other hand can provide the best privacy protection of all, since the interactions between the SP and IdP are mediated via the user's PC. Therefore the IdP does not need to know which SP the user is talking to.

#### 5.1.2.5. Phishing

OpenID is open to phishing attacks by either an evil SP or a third party attacker. The evil SP can pretend to redirect the user to their OP, but instead redirect them to their own masquerading OP site, in order to capture the user's login credentials. A third party attacker can entice the user to a masquerading SP site, and use a fabricated realm to further trick the user into believing they are accessing the genuine SP.

In Shibboleth, unlike OpenID, there is a deployment requirement, at least within the UK federation, that IdPs and SPs *must* make use of SSL in such a way that the user can (if they care) establish trust (on the basis of certificates issued by a trusted third party) that they are in fact dealing with the expected sites and not phishing sites. OpenID typically makes use of plain HTTP throughout, meaning the user has to rely on reading the address bar and trusting the local DNS. In practice, of course, users are likely to do neither, so there is also some susceptibility for the Shibboleth user. CardSpace on the other hand is much less susceptible to phishing attacks, since the user's PC acts as the discovery service and does not allow an evil SP to redirect the user to unknown and untrusted IdPs. The user's PC will need to be compromised first, which although not impossible to do, is much more complex to engineer than simply building a fraudulent OP site.

#### **5.1.2.6. Cross Site Request Forgery (CSRF)**

This form of attack relies on a third party attacker enticing the user to click on a URL whilst they are currently logged into their OpenID account (and hence have enabled single sign on to all OpenID enabled SPs). The page at the URL either contains a redirect to a SP site that the user has an OpenID account with, along with an embedded script that asks the SP to perform some request on behalf of the user, or contains embedded hidden iframes that silently contact the SP and request it to do the same thing, whilst the user is looking at the web page. At the Black Hat conference in 2007, Tsyurklevich demonstrated how money could be automatically debited from a bank account that supported OpenID authentication, using embedded iframes [7].

Shibboleth is not susceptible to this form of attack since the Shibboleth SP does *not* silently let the user in, but rather redirects the user to its WAYF service first, to ask the user which identity they wish to use. In this way, the user is alerted to the fact that they have been redirected to another service provider without their knowledge.

#### **5.1.2.7. Dependency on OpenID Provider**

Since the user must rely on the OpenID provider in order to gain access to all the SP sites, the user is dependent upon the OP running a 24/7 service 365 days a year, with no outages. This is an availability issue, and is a potential weakness in all centrally provided services, including Shibboleth. The difference here is that if the OpenID provider is providing a free service to the user, then the user has little leverage over them to improve availability. The cautious user will therefore enrol with multiple OPs to ensure continuity of service.

While a typical UK federation user is tied to the Shibboleth IdP of their home institution, this is rarely an issue as the institution tends to place high availability requirements on its IdP.

#### **5.1.2.8. Many Identity Providers, Fewer Service Providers**

There are many organisations who are willing to offer OpenID Provider services, and the number is growing almost daily. Big names such as Verisign, Google, IBM, Yahoo and Microsoft are committed to supporting OpenID [8] as identity providers. But currently there are fewer SPs who are willing to allow OpenID users to login to their services, and those that are, typically only provide low value or public services. There are many compelling reasons why this is so, which have already been discussed above.