D. Kemp
NSA

INTERNET-DRAFT
Expires 20 February 1997

August 13, 1996

The Public Key Login Protocol

<draft-kemp-auth-pklogin-01.txt>

1. Status of this memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress".

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

This Internet-Draft expires 20 February 1997.

2. Abstract

This document specifies the Public Key Login protocol, an ISO 9798-3 based security protocol which uses digital signatures to provide strong authentication for interactive sessions. It is suitable for use with unmodified session protocols such as Telnet and FTP, firewall proxies, dial-up Network Access Servers, remote mail servers, and similar applications. It provides functionality similar to One Time Passwords or handheld authentication tokens, and supports both one-way and mutual authentication. The protocol does not require the use of public key certificates and the associated infrastructure but can make use of them if they are available.

The Public Key Login protocol provides strong authentication at the beginning of a session and perhaps periodically thereafter, but does not provide communication channel integrity or confidentiality. Channel integrity is required in order to fully realize the benefits of strong authentication.

3. Introduction

Despite the development of communication protocols and mechanisms providing a full suite of security services (IPSEC, TLS, GSS-API), there remains a need for authentication mechanisms that can be used in environments with unmodified application and network protocols (Telnet, TCP/IP, PPP) and no security infrastructure (PKIX, DNSSEC, Kerberos, etc.). Plain password authentication is still widely used in these "traditional" environments because it is well understood and easy to implement. The critical weaknesses of plain passwords are also well

understood, and several approaches have been developed to provide stronger forms of authentication. Applications for strong authentication include:

* Firewall proxies.

* Modem connections to dial-up Network Access Servers, using authentication/authorization/accounting protocols such as [RADIUS] and TACACS+.

* Access to [IMAP] or POP remote email servers.

* Interactive sessions to individual hosts in a local environment, using a common authentication framework such as OSF Pluggable Authentication Modules [PAM].

Strong authentication methods designed specifically to be easy-to-deploy in traditional environments include:

* the One Time Password [OTP] protocol (once known as S/Key(TM)), which uses a one-way cryptographic hash function to compute passwords that cannot be intercepted and reused.

* various handheld authentication tokens (SecureID, SNK, SafeWord, Watchword), which use a shared secret key to either transform the time-of-day into a non-reusable password, or compute a response to a non-repeating challenge.

This document defines a third type of protocol which uses public key digital signatures to provide strong authentication in traditional environments. Like the alternatives, the Public Key Login (PKL) protocol is relatively easy to incorporate into existing systems, but it addresses some limitations of other strong authentication methods.

One Time Password systems must be initialized to allow a fixed number of logins. After that number of uses, the server must be reinitialized with a new seed before authentication can be used again. In addition, OTP relies on a user-generated secret which may be subject to guessing if not properly chosen; Steve Bellovin has described a system called "Monkey" that performs guessing attacks against OTP. OTP is subject to a race attack by an eavesdropper if the server does not take special precautions to disallow multiple simultaneous login attempts. An additional problem with OTP is that a server can issue a challenge from a different system, tricking the user into giving the server the ability to break into that other system. In contrast, the PKL protocol has no fixed login count, it uses cryptographic quality random private keys that are not subject to guessing, and it does not require special protection from eavesdroppers because a unique challenge is used for each login attempt.

Handheld authentication tokens rely on possession of a secret key as proof of identity. This does not provide non-repudiation - anyone able to obtain the secret key (by compromising a server, for example) is able to masquerade as the user to any servers using that secret key. Since user tokens contain only a small number of keys (typically from one to ten),

users wishing to authenticate to a large number of servers must either carry multiple tokens or share keys among multiple servers. The PKL protocol needs only one user key to authenticate to any number of servers. Each party has sole control of its own private key and the level of protection provided for it. If signature generation is implemented in a hardware token, the user's private key need not ever be exposed outside the token; this requires an attacker to take physical possession of a user's token in order to masquerade as that user.

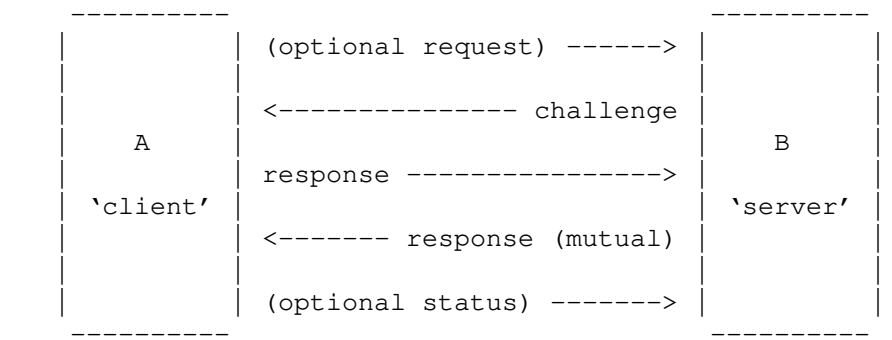
Neither OTP nor current handheld tokens allow the user to authenticate a server; the PKL protocol supports both unilateral and mutual authentication.

PKL can be implemented in either hardware or software, but it is particularly suited for use with portable cryptographic devices in the form of PC Cards, Smartcards, and Smartdisks. Such devices perform the processor-intensive digital signature generation and verification operations, and may provide better protection for a user's private key than can systems which generate digital signatures in software.

4. Protocol

The authentication exchange used in this protocol is defined by ISO 9798-3 [ISO9798] and the NIST Public Key Authentication Standard [PKAS]. This document specifies a particular instantiation of these standards, with a level of detail sufficient to enable the development of interoperable implementations.

Authentication is conducted between two entities, designated A and B. In the following discussion, entity A will be referred to as the client or user, and entity B will be referred to as the server.



The PKL authentication exchange consists of either two or three messages. The first two are used for unilateral authentication, in which B authenticates the identity of the A. All three messages are used for mutual authentication, in which each party authenticates the other. The messages are:

PKL1: The initial challenge from B to A
PKL2: The response from A to B
PKL3: The response from B to A in a mutual authentication exchange

In addition, optional messages may be used before and after the authentication exchange to provide additional unauthenticated (advisory) information. An optional authentication request message may precede the authentication exchange to indicate the client's desired protocol version, encoding, and public key access methods. The request message also includes an opaque data field which may be used to convey application-specific information such as a user/account identifier:

PKL0: Authentication request from A to B

After the authentication exchange, one of the following optional status messages may sent to return status information:

PKL4: Status message from B to A after unilateral authentication
PKL4: Status message from A to B after mutual authentication

4.1 Message Syntax

Each of the messages in a PKL authentication exchange consists of a message header, followed by one or more Tag-Length-Value (TLV) fields.

Each field in a message includes a tag that specifies the field type. Fields may also include a tag qualifier code and/or a field value and length as shown in the field definitions in Figure 1. The individual descriptions for each field type specify the applicable tag qualifier codes and field values for that type. The protocol may be extended by defining additional qualifier codes for existing types, or (if necessary) by defining new field types.

```
 %token MSG          /* Message label, for example 'PKL1'           */
 %token INT          /* Tag qualifier code (an integer)             */
 %token VALUE        /* Variable-length byte string and it's length */

 message : MSG ':' fields ':'
         ;

 fields  : field ':'      /* One or more fields per message         */
         | fields field ':'
         ;

 field   : 'M'            /* Mutual Authentication Flag             */
         | 'V' INT        /* Protocol Version Number                */
         | 'F' INT        /* Message Encoding Format                */
         | 'K' INT        /* Public Key Access Method               */
         | 'E' INT        /* Error/Status Reply Code                */
         | 'R' VALUE      /* Nonrepeating Value (nonce) from A or B */
         | 'S' VALUE      /* Digital Signature from A or B          */
         | 'C' INT VALUE  /* Entity ID or Cert Path for A or B      */
         | 'X' INT VALUE  /* Signed Opaque Data                     */
         | 'U' INT VALUE  /* Unsigned Opaque Data                   */
         ;
```

Figure 1. PKL Generic Message Syntax

Figure 1 defines the PKL generic message syntax using a yacc grammar. PKL messages may be encoded using either ASCII or binary formats. The message syntax for ASCII and binary formats is identical except that no field separator characters (':') are used with binary messages.

Figure 2 shows the fields included in each PKL message. Subscripts 'a' and 'b' are used with some field types to indicate to which entity the field value applies. For example, Rb refers to an R field (nonce) generated by 'B' (the server). Within the signature argument list, the field names indicate the order in which raw (unencoded) values are included in the signature calculation. The A: or B: in column 1 specifies the sender of the message.

Optional fields are enclosed in braces '{ }'; fields not identified as optional are required. Fields may appear in arbitrary order within a message, and no field except those specifically designated may appear more than once in a message. The optional authentication request message 'PKL0:' may include multiple fields of the same type to allow the client to specify a list of acceptable capabilities; the fields which may appear multiple times in that message are identified by an asterisk '*'.

```
------------------------------------------------------------------------
 A: Auth. Request:  { PKL0:{V}*:{F}*:{K}*:{U}:: }
 B: Challenge:        PKL1:K:Rb:Cb:{V}:{U}:{E}::
 A: Response:         PKL2:Ra:Ca:Sa(Ra,Rb,Cb,{X}):{U}:{X}:{M}:{E}::
 B: Mutual Response:  PKL3:Sb(Rb,Ra,Ca,{X}):{U}:{X}:{E}::
 B: Unilat. Status: { PKL4:E:: }
 A: Mutual Status:  { PKL4:E:: }
------------------------------------------------------------------------
```

Figure 2. PKL Messages

5. Message Fields

5.1 Protocol Version

This document describes version 1 of the Public Key Login protocol. The optional V field indicates the protocol version being used; if not present, version 1 is the default. Protocol evolution may be accommodated by defining additional field qualifier codes or field types without requiring a new version number to be assigned. Implementations should reject messages containing unrecognized field tags other than the allowed Opaque Data tags. If a server receives an authentication request message specifying a version that it does not support, it may either begin an authentication exchange with a supported version and including an explicit version field, or it may reject the request with an error reply code. The following version code is defined:

```
 V1:    PKL protocol version 1
```

5.2 Message Encoding Format

This document describes two message encoding formats, one ASCII and one binary. An optional Format field ('F') in the authentication request message (PKL0) selects the format of the following messages and enables the definition of additional message formats if necessary. If this field is not present, a default encoding suitable for the particular application is used. Message PKL0, if present, must use the F1 (ASCII) encoding. The following encodings are defined:

F1: ASCII message labels and field tags, Base64-encoded values
F2: binary message labels, field tags and field values

5.2.1 ASCII Encoding

In the ASCII format, each message consists of a message label, one or more tagged value fields, and a null field which serves as the message terminator:

PKL#:tag1-value1:tag2-value2:...::

The fixed-format message label ('PKL#') provides a marker for the beginning of authentication protocol messages. In many applications of this protocol, the authentication messages will travel over the same channel as other application-layer data. Implementations may need to scan a continuous datastream for message labels in order to locate and process the authentication messages.

Each field includes a tag type character, and may include a tag qualifier and/or a value as specified in Figure 1. Tag qualifiers are integers encoded as one or more ASCII digits. Field values are encoded using the [BASE64] mime-content-type and are separated from the tag by a dash '-'. Base64 encoding enables binary value data to be transported over ASCII-only channels, and it implicitly includes the length of the value. Fields are separated by colon characters ':', and the message terminator is represented by adjacent colons '::'. An actual message would look like:

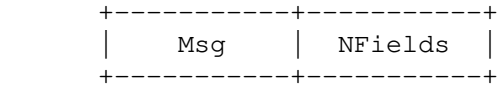
PKL1:K1:C0-cAQU5EUk:R-nZImJjnTNHJUtX3r::

Tags and message labels are ASCII constants and must be capitalized as shown. Whitespace is not allowed within message labels or field tags or between a tag and the following '-' character. Whitespace is allowed within values as specified in [BASE64]. Implementations must limit base64 values to lines of no more than 76 characters, and should break full PKL messages into lines of no more than 76 characters each.

5.2.2 Binary Encoding

Each message consists of a two byte message header followed by one or more variable length fields as shown in Figure 3. Unlike the ASCII encoding, no null field is used to terminate the message, instead a field count is included in the message header.

Message Header:



Fields:

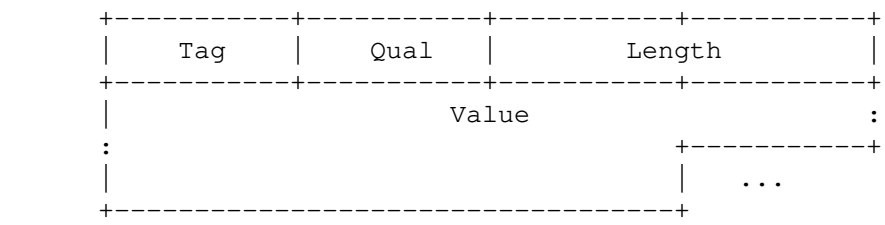


Figure 3. Binary PKL Message Encoding

The message header includes the message number and field count:

Msg Message label number - an ASCII digit '0'-'4' (0x30-0x34)
NFields Number of fields in this message.

The message fields immediately follow the header. Each field consists of a four byte descriptor and a variable length value:

Tag The ASCII Tag Type character ('M', 'V', 'F', etc.)
Qual Unsigned integer Tag Qualifier code in the range 0-254. If a qualifier is not used with the specified tag type, Qual shall have the value 255 (0xff).
Length The number of bytes in the following Value. This is a two byte unsigned integer, MSB first. If no value is used with the specified tag type, Length shall have the value 0.
Value A variable length binary value, MSB first, 0-65535 bytes

5.3 Public Key access method

This field has no associated value; the tag qualifier is used to select which messages contain entity ID or certificate fields. The entity which issues the challenge (referred to as the server) selects the key access method; client implementations must support any of the required methods. The client may request preferred method(s) using the optional authentication request message.

K1: The server ID is contained in message PKL1 and the client ID is contained in message PKL2. The IDs correspond to the personality (private key) used by the signer, and are used by the verifier to retrieve the signer's public key or certificate.

K2: The server's certificate is contained in message PKL1 and the client's certificate is contained in message PKL2. The entity IDs, if required for access control, are derived from the contents of the certificates (Subject Distinguished Name, or Issuer/Serial Number, etc).

K3: The server's and client's full certificate paths are contained in messages PKL1 and PKL2. In addition to requiring a large amount of data to be exchanged, this method potentially requires the parties to agree at runtime on a path root (common point of trust). This protocol does not define any method of negotiating desired Certificate Authorities.

K4: The server's and client's full certificate paths and auxilliary information (such as Certificate Revocation List (CRL) and Compromised Key List (CKL)) are contained in messages PKL1 and PKL2.

The following table summarizes the ID/Certificate fields contained in each of the messages as specified by the Key Access Method, and the methods which must be supported by client implementations:

	PKL1	PKL2	PKL3	Required?
K1:	B	A	-	Yes
K2:	Cb	Ca	-	Yes
K3:	Cbpath	CApath	-	No
K4:	Cbpath+	CApath+	-	No

5.4 Nonces

The Ra and Rb fields are non-repeating values generated by the client and server respectively. The protocol definition does not specify the size or content of these fields because such requirements are not necessary to achieve interoperability. However, the security of the protocol depends critically on the quality of the data in these fields. See the discussion below under Security Considerations.

5.5 Identifiers and Certificates

The PKL protocol requires the use of public/private key pairs and associated parameters to generate and verify digital signatures, but does not specify how implementations are to maintain this information. In the simplest case, keys and parameters are statically configured on the client and server. This is equivalent to configuring a One Time Password system or a symmetric key authentication token. Alternatively, public key certificates can be statically configured, transmitted within the PKL protocol, or retrieved from a directory server.

5.5.1 Certificate Retrieval

When using the ID-based Key Access option (K1), implementations use an entity ID to refer to a public key or certificate. Certificates (X.509, SPKI, PGP, etc) are exchanged once using a registration protocol described below, and then only entity IDs need be transmitted during authentication. The PKL entity ID is defined to be a hash of the entity's certificate. This allows the entity ID to be uniquely determined from the

certificate contents and vice versa. The public key is included in the hash to provide uniqueness; the rest of the certificate contents are included because for some applications it may be necessary to distinguish between multiple certificates containing the same public key.

For unilateral authentication, entity B is not required to have a public/private key pair or certificate. In this case B is referred to by an entity name (certificate type C0). The name should be meaningful to a human user, but is not used or interpreted by the protocol.

Implementations may maintain a mapping between entity IDs and certificate IDs meaningful to a directory service (Distinguished Names, Issuer/Serial numbers, etc.) to allow retrieval of certificates and validity information from the directory using a protocol such as [LDAP]. Implementations may also maintain a mapping between entity IDs and userid's or other application-specific information to be used for access control. Entity IDs transmitted in protocol messages may be truncated for transmission efficiency - the only requirement is that the ID be unique among those known to the receiver. For many applications the first 48 bits of the entity ID will be sufficient to uniquely identify the entity's certificate.

5.5.2 Certificate Transmission

When using certificate-based Key Access options (K2,K3,K4), public key certificates are transmitted as part of the authentication exchange. In this case, implementations are able to verify that the binding between subject ID and public key is valid by validating all certificates in the path between a trusted party and the entity being authenticated.

5.5.3 Certificate Type Assignments

C0	Entity ID - name (no certificate)
C1	DER-encoded X.509 certificate(s) [X500]. Because the DER encoding is self-delimiting, multiple certificates may simply be concatenated for use with the K3 and K4 (certificate path) Key Access options. The first certificate must be that of the entity to be authenticated.
C2	PGP [PGP]
C3	Secure DNS [DNSSEC]
C4	MD5 of Unsigned DH Public Value [UDH]
C5	MD5 of Unsigned Elliptic Curve Public Value
C6	MD5 of Unsigned RSA Public Value
C7	(unused - X.509 CRLs are included with type C1)
C8	SPKI certificate(s) [SPKI]
C9	Entity ID - SHA Hash of certificate [SHA]
c10	Entity ID - MD5 Hash of certificate

Certificate Type Values 0 through 10 are assigned as shown above. These assignments are compatible with those given in [CDP]. Values 11 through 249 inclusive are reserved to the IANA for future allocation as Assigned Numbers. Certificate Type Values 250 through 255 inclusive are available for private use.

5.6 Signed and Unsigned Opaque Data

The 'X#' and 'U#' fields represent signed and unsigned data which is not interpreted by the PKL protocol, and which can be used for application-specific purposes. The tags X1: - X127: and U0: - U127: are reserved to the IANA for future allocation as Assigned Numbers; implementations of this version of the protocol must not generate reserved field tags and must fail to authenticate if a reserved field tag is received. Implementations may use non-reserved tag codes (128-255) for any purpose by private agreement. Unrecognized non-reserved tag codes should be ignored on receipt.

Signed data is transmitted in the 'X' field and included in the digital signature calculations. It is thus protected from modification and bound to the sender's authenticated identity. The PKL protocol makes no use of this field except to guarantee it's integrity and authenticity. This field might be used, for example, with a key establishment protocol to bind a session key to authenticated endpoint identities. The tag X0: is reserved for use with key agreement information such as Diffie-Hellman values.

Unsigned data is transmitted in the 'U' field but is not used in any way and is not protected from modification by the protocol. Implementations must not rely on unsigned data in any way that affects the success or failure of authentication or affects any actions based on the authenticated identity. The unsigned data field is included in the protocol solely to accommodate unforeseen applications, and should be used only after thoughtful analysis.

5.7 Signatures

Digital signatures are calculated over the fields listed in the argument list, in the specified order. For example 'Sa(Ra,Rb,Cb,{X})' specifies the following processing steps to be performed by A:

* Concatenate the required values Ra, Rb, and Cb in this order

* Append the optional signed data X#, if present in the message

* Calculate the digital signature

* Base64-encode the signature value (for ASCII-format messages)

To achieve interoperability, signatures must be calculated using the specified argument order. Note: there is no particular significance to the order in which nonces appear in the signed data - (Ra,Rb) in message PKL2 vs. (Rb,Ra) in message PKL3; it is defined this way solely for consistency with [ISO9798].

The algorithm used to generate and verify digital signatures is specified in the entity's public key certificate, or in equivalent configuration data if certificates are not used. The value of the signature ("S") fields depend on the algorithm used:

Digital Signature Standard: The [DSS] signature of a message is a pair of bit strings designated R and S. The value of the signature field is R followed by (concatenated with) S. The DSS public key (community parameters P, Q, and G, and user-specific public value Y) may be extracted from a certificate, manually configured, or obtained by other methods outside the scope of this protocol. P, Q, and G may be inherited from a parent if they are not present in an entity's certificate.

RSA Signature: The RSA signature of a message is a single bit string designated S, which is the value of the signature field. The RSA public key (modulus N and public exponent E) may be extracted from a certificate, manually configured, or obtained by other means.

5.8 Mutual Authentication Flag

The optional 'M' field in message PKL2 allows the client to request mutual authentication. If it is present, the server must generate message PKL3 if the server is configured to allow mutual authentication. If the server cannot generate message PKL3, it should return an error indication (reply code or human-readable message). If the 'M' field is not present, the server must not send message PKL3. Note that there is no reason to allow the server to request mutual authentication, since the server cannot force a client to authenticate any information sent by the server. This field has no tag qualifier code or associated value.

Implementations of the PKL protocol should support both unilateral and mutual authentication.

5.9 Reply Codes

This optional field has no associated value; the tag qualifier contains the acknowledgment or error status code. This field is not sent by default, but if a reply code field is sent in one authentication message, all following messages in the current exchange must include reply codes. In this manner either client or server can request that reply codes be used. Implementations must support reply codes for Authentication Success, Authentication Failure, and Continue. Implementations may support additional reply codes which provide more detailed information.

In environments where the authentication exchange is visible to the user, it is generally obvious that an authentication request has either succeeded or failed. Additional information can be communicated in free form outside the authentication protocol, and reply codes may not be needed. In other environments, such as modem dialer chat scripts, it may be useful to transmit status information in machine-readable form using this field.

Reply codes are based on the three digit format defined in [FTP]. The first digit is a '2' for success/continue and '5' for failure. The second digit is a '0' for protocol syntax status and a '3' for authentication status. The third digit gives additional detail.

For the binary message format, the three digit code is converted to a single-byte quantity. The most significant bit (0x80) represents the first digit: 0 for success and 1 for failure; the remaining 7 bits encode the binary representation of digits 2 and 3.

The following reply codes, shown for both ASCII and binary formats, are defined:

Required:

E200 0x00: NOP - Continue. This code is sent by either the client or server to request that reply codes be used in the following messages, or if no other status is applicable to the current message.
E230 0x1E: Authentication succeeded.
E530 0x9E: Authentication failed.

Optional:

E201 0x01: Info - Unrecognized non-reserved field tag.
E500 0x80: Syntax error - unrecognized or invalid field tag.
E501 0x81: Base64 value decoding error.
E502 0x82: Requested protocol version "V#" not supported.
E503 0x83: Requested message encoding "F#" not supported.
E504 0x84: Requested key access method "K#" not supported.
E505 0x85: Mutual authentication not supported.
E531 0x9F: Invalid entity ID - public key could not be retrieved.
E532 0xA0: Non-unique entity ID - retry with a longer ID.
E534 0xA2: Invalid public key certificate - many possible causes, including expired or revoked certificate, path validation failed, unknown or unsupported certificate format, etc.

6. Registration

The PKL protocol allows certificates to be either passed within the authentication exchange or referenced by an entity ID. The latter case generates considerably less traffic during authentication, but requires that identifiers be assigned to certificates or public keys before the authentication protocol can be used. This section specifies a registration protocol by which entity IDs can be assigned to certificates.

The registration protocol consists of two messages, which may be exchanged offline (via email, file transfer, floppy disk, etc.). The messages are protected from modification by a secret key, which should be communicated separately from the registration messages. The registration messages contain the Certificates and a keyed signature field to provide integrity protection. They are encoded using a format similar to ASCII authentication messages, but with a different message header 'PKR:'. The client's and server's messages contain the same field types, and may be exchanged in either order.

B: PKR:Cb:Sb(K,Cb)::
A: PKR:Ca:Sa(K,Ca)::

Registration consists of the following steps:

1. One of the parties (either A or B) generates a registration key K and sends it to the other party. It is recommended that K be sent over a different channel than the registration messages, for example by telephone, fax, mail, or courier service. Implementations must allow K to be specified in hexadecimal format, and should also allow K to be entered in a more user-friendly passphrase format. If a passphrase is used, it must be canonicalized by removing all whitespace and translating all lower case characters to upper case.

2. The server B forms its full certificate path Cb beginning with B's certificate and ending with the certificate of an entity trusted by both A and B. B then creates a registration message containing Cb and B's signature of K and Cb, and sends the message to A.

3. The user A generates an analagous registration message using K and Ca and sends it to B. The two registration messages may be sent in parallel or in either order.

4. The server B processes the message from A by validating the certificate path Ca, prompting the operator for K, verifying A's signature of K and Ca, and computing A's entity ID. B then stores either Ca itself, or information sufficient to retrieve Ca from a directory, indexed by A's entity ID.

5. A processes the registration message from B in a similar manner, storing the relationship between B's entity ID and Cb.

7. Examples

A minimal unilateral authentication exchange might look like:

```
B:  PKL1:K1:C0-cAQU5EUk:R-nZImJjnTNHJUtX::
A:  PKL2:R-Asho84kLN3/IJm:S-Wiy6IesKvjL5rL9WjXUb9BkAmw=::
```

A mutual authentication exchange with status codes might look like:

```
B:  PKL1:K1:C0-cAQU5EUk:R-nZImJjnTNHJUtX::
A:  PKL2:R-Asho84kLN3/IJm:S-Wiy6IesKvjL5rL9WjXUb9BkAmw=:E200::
B:  PKL3:S-6IesKvjL5rL9WjXUb9MwT9bpObY=:E230::
A:  PKL4:E230::
```

If certificates are exchanged, the messages are considerably longer.

The Internet Message Access Protocol [IMAP4] includes an AUTHENTICATE command which could be used to invoke PKL authentication. Because IMAP4 authentication messages are encoded as single Base64 strings, it is appropriate to use the PKL binary message format followed by Base64 encoding. A PKL exchange within IMAP4 might look like the following: (C: and S: indicate lines sent by the client and server respectively.)

```
S:  * OK IMAP4 Server
C:  A001 AUTHENTICATE PKL0:F2::
S:  + xxxxxxxxxxxxxxxxxxxxxx
C:  yyyyyyyyyyyyyyyyyyyyyy
S:  A001 OK PKL authentication successful
```

8. Implementation Experience

A protocol similar to this one, the "Mosaic Authentication Protocol" (MAP), has been in use since April 1994 to provide network and dial-up access to computers at NSA, and has been commercialized in the LOCKout(TM) product from Secure Computing Corporation. MAP was also the basis for HTTP-Fortezza, a prototype demonstration of strong access control for Web applications.

The PKL protocol is a generalization of MAP, designed to comply with the NIST Public Key Authentication Standard [PKAS] and to address specific problems identified during operational use of MAP. No implementations of PKL currently exist, but at least two are under development. All Government applications of MAP are expected to migrate to PKL by the effective date of the NIST Authentication Standard (currently TBD).

9. Security Considerations

9.1 Authentication, Integrity, and Confidentiality

The primary limitation of authentication-only protocols, including this one, is that they provide no protection to the data exchanged over the connection. Authentication by itself does not protect user data from an eavesdropper, does not protect the user's session against active attacks [CERT], and is of limited usefulness when the channel between authenticating parties is not trusted. If a protocol providing both authentication and channel integrity protection ([IPSEC], [TLS], etc.) is available, it should be used in lieu of an authentication-only protocol. In some situations it may be useful to combine an authentication-only protocol with other methods, for example to provide user authentication at the application layer over (host-keyed) Virtual Private Networks using IPSEC at the network layer.

9.2 User Interaction

Like the OTP and handheld token authentication methods, the Public Key Login protocol is intended to be used on behalf of a human user. User agent (client) software should only operate in response to a user-initiated connection request, and should display the identity of the server and request confirmation from the user before signing and sending a response message. The client software should not respond to unsolicited challenges or generate response messages without positive confirmation from the user.

9.3 Challenge Response vs. Time

The Public Key Login protocol uses an authentication method based on signed challenges. This avoids the following problems faced by public key authentication methods (for example, [X.509]) based on signed time fields:

* Clocks must be synchronized. This may be difficult to achieve in some situations.

* Even environments which provide tightly-controlled clocks (using e.g. NTP), the security of the authentication protocol depends on the security of the time synchronization method. A protocol which does not have this critical reliance on an external service is preferable.

* Even in environments which provide synchronized clocks which are resistant to failure or attack, time-based authentication methods must allow a window of acceptance to accommodate communication delays. Eliminating vulnerability to replay attacks within this window requires additional complication, particularly when multiple servers are involved.

In contrast, challenge-response authentication is only subject to replay attacks if a challenge is repeated. This is easily prevented, as addressed in the following section.

9.4 Nonces

The ability of challenge-response protocols to protect against replay attacks relies on the assumption that nonces will not be repeated during the lifetime of the digital signature keys with which they are used. Random numbers are often used as nonces. However, a number chosen at random will have a small but non-zero probability of taking any particular value. The probability of repeating one of the previously-used nonces increases with continuing use of the authentication protocol with a particular signature key. Nonces containing a deterministic component such as time or a sequence number have zero probability of repeating as long as the deterministic component is reliable, but achieving the required level of reliability is problematic. Accordingly, it is recommended that implementations of this protocol construct nonces by concatenating a random value with a long-term non-repeating value such as time. Note that this application of a time value does not require that clocks be synchronized either with real-world time or between the authenticating parties. If a single user authenticates to multiple servers and the servers' clocks are not synchronized, then any time component of a nonce does not contribute to replay protection for the server with the later clock value. Therefore, implementations of this protocol should construct nonces with a random component sufficiently large to provide adequate protection without relying on a non-repeating component.

It is recommended that nonces contain a minimum of 64 bits (8 bytes).

9.5 Authentication, Identities and Authorization

The PKL protocol provides authentication - proof that an entity at the other end of a connection possesses the private key corresponding to a given public key. Certificates are used to attach authorizations, either explicitly, or implicitly by use of an entity name, to the public key. Representation and automated processing of authorizations is the focus of current research efforts, for example [SPKI]. Mechanisms such as Access Control Lists (ACLs) based on users and groups may also be used to grant specific authorizations to specific entities.

Note that implementations must take care to ensure that access control or other actions are based on the identity which has been authenticated, not on claimed or otherwise unauthenticated identities. This is particularly important when certificates are contained in the protocol messages, because that implies that the manual registration procedure was not used to establish the authorizations applicable to a particular certificate. For example, a user attempting to login with username "Bob" could present a valid certificate for user "Fred" and a signature generated with Fred's private key. After successfully verifying the signature, the server must not log the user on to Bob's account.

9.6 Certificate Path Validation

This protocol is usable with a wide range of keying methods, from static keys to certificate transmission to full online directory services. If locally-generated keys or certificates are used, validation is likely to be less complex than if certificates are obtained from a large-scale PKI. In general, validation may involve verifying certificate paths back to a trusted authority whose public key is known locally, and verifying that the validity period of all certificates in the path include the current date. Validation may also involve retrieving and checking revocation lists to ensure that no certificates in the path have been revoked. Certificate path validation procedures do not affect interoperability of the PKL protocol itself, but appropriate validation procedures are critical to assuring that names are bound to the proper keys.

10. References

[BASE64] Internet RFC 1521, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", Section 5.2, "Base64 Content-Transfer-Encoding", Borenstein & Freed, September 1993

[CDP] Internet Draft (work-in-progress), "Certificate Discovery Protocol", A. Aziz, T. Markson, H. Prafullchandra, R. Skrenta, G. Caronni, draft-ietf-ipsec-cdp-01.txt

[CERT] Computer Emergency Response Team, "IP Spoofing and Hijacked Terminal Connections", CA-95:01, January 1995, ftp://info.cert.org/pub/cert_advisories/CA-95:01.IP.spoofing

[DNSSEC] Internet Draft (work-in-progress), "Domain Name Security Extensions", D. Eastlake, C. Kaufman, draft-ietf-dnssec-secext-04.txt

[DSS] NIST FIPS Pub 186, "Digital Signature Standard (DSS)", 19 May 1994, http://csrc.ncsl.nist.gov/fips/fips186.txt

[FTP] Internet RFC 959, "File Transfer Protocol", Section 4.2, "FTP Replies", Postel & Reynolds, October 1985

[IMAP4] Internet RFC 1930, "Internet Message Access Protocol - Version 4", M. Crispin, December 1994. See also RFC 1931, IMAP4 Authentication Mechanisms", J. Myers, December 1994.

[ISO9798] ISO/IEC 9798-3, "Information technology - Security techniques - Entity authentication mechanisms - Part 3: Entity authentication using a public key algorithm", Section 5.1.2 "Two pass unilateral authentication" and Section 5.2.2 "Three pass mutual authentication", 1993

[LDAP] Internet Draft (work-in-progress), "Lightweight Directory Access Protocol (v3)", M. Wahl, T. Howes, S. Kille, draft-ietf-asid-ldapv3-protocol-01.txt, 5 June 1996. See also RFC 1777, RFC 1798

[OTP] Internet RFC 1938, "A One-Time Password System", N. Haller, C. Metz, May 1996

[PAM] OSF RFC 86.0, "Unified Login with Pluggable Authentication Modules (PAM)", October 1995 http://www.pilgrim.umass.edu/pub/osf_dce/RFC/rfc86.0.txt

[PGP] Internet Draft (work-in-progress) "PGP Message Exchange Formats", D. Atkins, W. Stallings, P. Zimmermann, draft-atkins-pgpformats-01.txt

[PKAS] NIST FIPS Pub JJJ, "Entity Authentication using Public Key Technology", Draft 29 March 1996 http://csrc.ncsl.nist.gov/fips/fipsjjj.txt

[RADIUS] Internet-Draft (work-in-progress), "Remote Authentication Dial In User Service (RADIUS)", November 1995, draft-rigney-radius-01.txt

[SHA] NIST FIPS Pub 180-1, "Secure Hash Standard", 31 May 1994, http://csrc.ncsl.nist.gov/fips/fip180-1.txt

[SPKI] Internet-Draft (work-in-progress), "Simple Public Key Certificate", Ellison, Frantz, Thomas, draft-xxx

[UDH] Internet Draft (work-in-progress), "Encoding of an Unsigned Diffie-Hellman Public Value", A. Aziz, T. Markson, H. Prafullchandra, draft-ietf-ipsec-skip-udh-00.txt

[X.500] ITU X.500, "The Directoy"

[X.509] ISO 9594-8 / ITU X.509, "The Directory - Authentication Framework", (1988 version: http://sunsite.doc.ic.ac.uk/computing/ccitt/ccitt-standards/1992/X/x509_1.asc.gz)

11. Acknowledgements

Thanks to Hugh Harney of SPARTA, Inc., Dave Balenson of Trusted Information Systems, and Dale Hapeman of Booz-Allen-Hamilton for their reviews of this draft and many constructive suggestions. Thanks to Carl Ellison of Cybercash for suggesting the ideas embodied in the registration protocol. Special thanks to Jim Press of ICL for suggesting several changes to make the protocol more usable in a real application.

12. Author's Address

David P. Kemp
National Security Agency, X31
9800 Savage Road
Fort Meade, MD 20755
USA

Email: dpkemp@missi.ncsc.mil