Security analysis of the RFB 5.0 protocol

Nicholas Wilson
RealVNC Ltd
`security@realvnc.com`

Version 1.0	10 July 2015	Original version for VNC Developer SDK 1.0.0
Version 1.2	9 April 2018	Updated for VNC Connect 6.0.0
Version 1.3	24 May 2018	Addition of RA4b and RA4-PSK-Only cipher suites for Instant Support 1.0.0

Abstract

In RFB 5.0, a new security flow is introduced which supersedes previous analysis done on the security of the RFB protocol. This document aims to describe the general properties of RFB 5.0, to provide a security policy in which to describe attacks on the protocol, and to provide a detailed analysis of the properties provided by the "RA4" cipher suites. The encodings used on the wire are not covered by this document.

Contents

1 Introduction to RFB 5.0

The Remote Frame-Buffer (RFB) protocol, version 5.0 is an updated version of RFB 4.1 containing changes to the setup stages of the protocol. The initial portions of the protocol contain generic handshaking messages used to establish a secure and authenticated connection between a server and a viewer, after which the clients exchange keyboard and pointer events and display updates. In this document, the latter 'normal' stages of the protocol do not concern us, and we consider only the cipher suite phase of the protocol. These phases are shown in Figure 1.

The RFB protocol opens with the *handshaking phase*, the purpose of which is to agree upon the protocol version. The client acting as the server sends its highest-supported RFB version, and the viewer responds with its chosen RFB version. During the *cipher suite phase* after this, the endpoints negotiate a cipher suite, then the viewer authenticates to the server in the *authentication phase*. The further phases of the protocol (defined in the RFB 5.0 specification) are an *initialisation phase* where the client and server exchange extension messages ending with a ServerInit/ClientInit exchange, and finally the *normal phase* of protocol interaction using input and framebuffer update messages.

The cipher suite allows the server and viewer to set up an encrypted channel. The server authenticates to the viewer using an RSA key for identification in all the ciphers defined, and at the end of the cipher phase the viewer has established and verified the server's identity. The viewer is generally still anonymous at the end of the cipher phase, because authentication of the viewer to the server is covered by its own protocol phase.

In the authentication phase, the server prompts the viewer to provide credentials and then grants access. The majority of VNC customers use "system authentication", in which the server prompts the viewer for a UNIX or Windows username and password which are verified using the Local Security Authority (Windows) or PAM (UNIX). Kerberos tokens may also be used. The details of the authentication phase are not covered by this protocol analysis document.

Although the cipher suite is negotiated to allow for forward-compatibility in the protocol, each product produced by RealVNC in fact has very a limited selection of ciphers, providing minor differentiation between products, and a single overall design is used. This cipher is "RA4" ("RSA with AES version 4"), an updated version of the RA2 security type developed ten years ago for RFB 4.0.

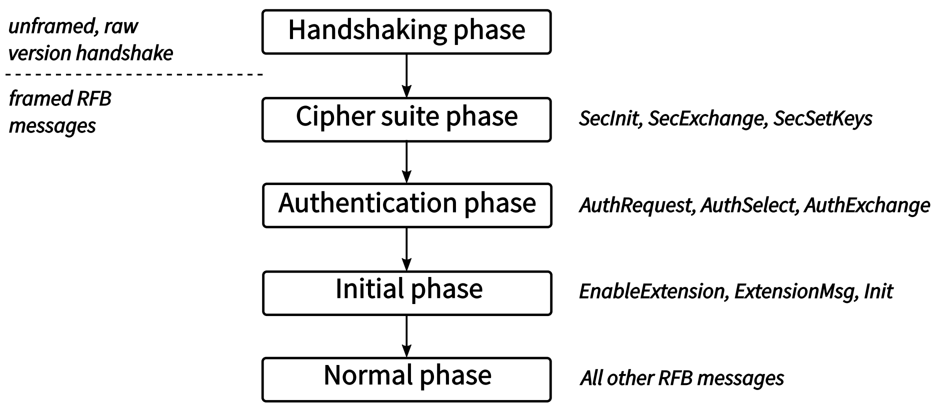
unframed, raw version handshake
framed RFB messages
Handshaking phase
Cipher suite phase
SecInit, SecExchange, SecSetKeys
Authentication phase
AuthRequest, AuthSelect, AuthExchange
Initial phase
EnableExtension, ExtensionMsg, Init
Normal phase
All other RFB messages

Figure 1: Phases of RFB, showing the message types exchanged in each phase

Cipher suite	*Authentication phase*	*Initialisation and normal phases*
RA4ne-128	AES-128	plaintext with HMAC
RA4-128, RA4-PSK-128, RA4-PSK-128-Only	AES-128	AES-128
RA4-256, RA4b-256, RA4-PSK-256, RA4b-PSK-256, RA4-PSK-256-Only, RA4b-PSK-256-Only	AES-256	AES-256

Table 1: Summary of message encapsulation used during the phases of RFB 5.0. For AES encapsulation, GCM authentication mode is used.

Cipher suite	Diffie–Hellman group	RSA?	PSK?	Hash
RA4-128, RA4ne-128	curve25519	Yes	No	SHA-256
RA4-256	curve25519	Yes (High)	No	SHA-512
RA4b-256	curve448	Yes (High)	No	SHA-512
RA4-PSK-128	curve25519	Yes	Yes	SHA-256
RA4-PSK-256	curve25519	Yes (High)	Yes	SHA-512
RA4b-PSK-256	curve448	Yes (High)	Yes	SHA-512
RA4-PSK-Only-128	curve25519	No	Yes	SHA-256
RA4-PSK-Only-256	curve25519	No	Yes	SHA-512
RA4b-PSK-Only-256	curve448	No	Yes	SHA-512

Table 2: Summary of key exchange methods used. For RSA key echange, OAEP is used to exchange nonces; a minimum key size of 1024 bits is enforced, or 2048 bits where marked “High”.

The variants of RA4 provide different features for different product lines: RA4-128 is the main variant of RA4 and uses AES-128. RA4-256 is identical except for the use of AES-256, allowing it to be provided in cases where the additional encryption strength is required, along with RA4b-256 using an updated choice of elliptic curve. Also, there is RA4ne-128 for use in secure environments where the cost of encryption is prohibitive, and which is a no-encryption variant using AES-128 to protect the authentication phase, but then switching down to plaintext-with-HMAC; it is identical to RA4-128 for the purposes of the key exchange analysis in section 4. There are three further variants using an additional Pre-Shared Key, RA4-PSK-128, RA4-PSK-256, and RA4b-PSK-256, which are used to provide additional security for connections brokered via VNC Cloud (described below), and three variants RA4-PSK-Only-128, RA4-PSK-Only-256, and RA4b-PSK-Only-256 for VNC Cloud connections under the Instant Support brand.

Finally, the Diffie–Hellman key-exchange used in RA4 may be performed over different groups. The original RA4 cipher suites use curve25519, and the later-released RA4b versions use curve448 (see section 4.3 for details). These variants are summarised in Tables 1 and 2.

2 Introduction to VNC Cloud

VNC Cloud is an collection of web-services providing a public address book and connection brokering service to clients who may not otherwise have a public address on the Internet. For example, a domestic computer may change its public IP address frequently, and a computer on an office intranet will use NAT (Network Address Translation) to access the Internet, and has no external address on which to receive incoming connections.

VNC Cloud provides a persistent long-lasting Cloud address to computers, enabling a viewer to initiate a connection to a server, without needing to know the server's current IP address. Through a connection brokering service, the viewer can perform a handshake with the server, and attempt NAT hole punching (STUN). Finally, if both parties are behind restrictive NAT devices such that a direct connection is not possible, they may use the VNC Cloud relay service to exchange data.

Through the VNC Cloud connection broker, the viewer and the server exchange nonces along with their RSA public keys. Then, during the subsequent RFB session the RSA keys and nonces from the used in the RA4-PSK cipher suites are used to provide additional authentication of the peer.

3 Overall security policy

In RFB, the data is typically sent across an unsecured network, so we assume that an attacker may drop, alter, or craft messages to either the server or viewer at any time. An RFB cipher suite must be able to detect such tampering to allow the implementation to terminate the connection. We do not regard denial-of-service as a threat we guard against, since an attacker with access to the network in between two parties can certainly cut the cable or selectively drop packets and cause the connection to fail.

We consider the attacker to be "external" to the two RFB peers, having access only to data which is sent out over the network. At any moment, the attacker has access to all messages that either party has previously attempted to send, but cannot sample hidden state internal to the peer. Our policy model therefore does not cover attacks from a subverted operating system, for example, which is able to read the contents of any process's memory at will. In practice, these assumptions widely hold for many typical threats, and it is out of the scope of a networking protocol to attempt to defend against a more powerful attacker.

We assume that any private keys and nonces used are not known initially to the attacker (although they would become known if sent out over the network as part of the protocol). The security policy requires that private keys and nonces be kept secret for ever (or equivalently destroyed after use). Where private keys are used for identification, no protocol can distinguish between a genuine peer and an attacker with a copy of the peer's private identification key.[1]

[1] The Key Compromise Impersonation property describes an extended model including a second sort of attacker, who has read-only access to a private key but has not gained access to any other resources on the system. A protocol with resistance to KCI prevents such an attacker from impersonating other parties to the partially-compromised peer (which could elevate the leak of the private key to providing access to other resources on the system). In the context of the RFB 5.0 cipher suites however, key compromise of either server or viewer keys is already fatal, so although the RA4 cipher suites are in fact designed with KCI resistance, we do not include this scenario in our analysis.

Additionally, we grant that the attacker is able to access the private key K^{-1} after the legitimate peers have finished exchanging messages. In order to operate under this security policy, the cipher suite must provide Perfect Forward Secrecy, so that compromise of long-term identification keys does not compromise previous sessions. The model's restriction that the legitimate peers must have finished sending and reading messages means that the compromised server key cannot be used in active attacks to impersonate the server to the viewer, which is out of the scope of the protection provided by PFS.[2]

Finally, our policy model generally assumes that both parties are following the protocol correctly. In practice, it is unlikely that real implementations are perfect, but we do not attempt to consider all the possible implementation errors and analyse the resulting weaknesses. The threats we consider therefore concern perfect peers. In particular, the analysis depends on peers not sending or receiving messages which do not form part of the protocol, so that we only need to reason about the messages in the protocol to determine whether or not a peer provides an attacker with a decryption or signing oracle. Similarly, peers may not send out over the network any secret material such as keys or nonces except where this explicitly forms part of the protocol.

4 Analysis of RA4

4.1 Security policy for RA4

As well as implementing the overall security policy for RFB 5.0 above, the policy must describe the specific keys used in the RA4 exchange. Each RA4 endpoint uses two keys, an RSA keypair and pair of Diffie–Hellman public and private values.

The RSA keypair is used to attest to a challenge sent sent by the peer, to authenticate each party in the key exchange.

The key used by the viewer K_V is typically not known to the server, nor does the server perform any checks on K_V, since the client subsequently authenticates using a UNIX or Windows password; however, in some deployments of the protocol it may be used to provide a long-term identity for the viewer. We assume here that it is ephemeral and hence unknown to the server.

The server's RSA public key K_S is used by the viewer to perform strict checking of the server identity. Management of the server keys is performed out-of-band. In the discussion that follows, the server public key is assumed to be well-known, so that V knows when connecting to "`server63.example.com`" (for example) the exact public key K_S that the server must present.[3] If a different key is presented, or if the RFB peer fails to provide proof that it possesses the private key K_S^{-1}, then the protocol provides feedback to the viewer to abort the connection before any sensitive information is disclosed.

[2]In practice, this models a very useful property where a key is compromised, and we desire to know what effect this has on previous sessions. For example, an employee may have edited some documents over VNC from his phone. If the phone is stolen by an attacker who has recorded the session, although the attacker is able to extract the long-term private key from the phone, the recorded session still cannot be decrypted.

[3]The well-known server key can be distributed in various ways. VNC Viewer follows the "Key Continuity Model" for key management, comparing K_S against a list of previously-seen server identities, or else prompting the user to approve keys manually for the first connection to each server. In both cases VNC Viewer is able to validate the server key.

For each connection V and S also possess a pair of Diffie–Hellman public and private values (J_V, J_V^{-1}) and (J_S, J_S^{-1}) respectively, the private values not known to the attacker.

4.2 RA4 messages

The viewer and server begin by performing a RFB version negotiation, for compatibility with legacy and future clients, where RFB_S and RFB_V are the offered and chosen RFB versions. In this example, the viewer indicates support for the RA4 cipher by including it in the *CipherList* data. Our analysis covers the case where the server chooses *Cipher* to be RA4. The 32-byte cookies $Cookie_V$ and $Cookie_S$ must be chosen such that no pair $(Cookie_V, Cookie_S)$ is reused for any two connections to the same server, and such that an attacker is unable to force use of a particular value for $Cookie_S$; in practice, random values are acceptable since the probability of a collision is negligible.

$$S \rightarrow V : RFB_S \tag{RFB-1}$$
$$V \rightarrow S : RFB_V, Cookie_V, CipherList \tag{RFB-2}$$
$$S \rightarrow V : Cookie_S, Cipher = \text{“RA4”} \tag{RFB-3}$$

The RA4 exchange consists of an exchange of the Diffie–Hellman public values, along with a challenge-response identification protocol using the RSA keys:

$$S \rightarrow V : J_S, K_S \tag{RA4-1}$$
$$V \rightarrow S : J_V, K_V, \{N_V\}_{K_S} \tag{RA4-2}$$
$$S \rightarrow V : \{N_S\}_{K_V} \tag{RA4-3}$$

At this stage, (N_V, N_S) is a secret known only to the owners of K_V^{-1} and K_S^{-1}, but the messages so far have not involved use of the private keys, so there is no proof yet that the private key owners have taken part in the protocol exchanges.[4] In order to fully authenticate the exchange, a verification digest is sent in each direction, wrapped using the cipher's authenticated encryption method:

$$S \rightarrow V : \{Digest_{SV}\}_{SymKey_{SV}} \tag{RFB-4}$$
$$V \rightarrow S : \{Digest_{VS}\}_{SymKey_{VS}} \tag{RFB-5}$$

The RFB-4 and RFB-5 messages and all subsequent RFB protocol protocol data are exchanged in encrypted and authenticated packets containing the ciphertext and a MAC.

The digests and keys used in the messages above rely on a family of Pseudo-Random Functions $\text{PRF}_P(SourceMaterial, Context)$ which hash together source material and associated context, and produce a stretched output of the desired length. The PRFs specified by each value of the parameter P produce independent output. The values for $Digest_{SV}$, $Digest_{VS}$, $SymKey_{SV}$, and $SymKey_{VS}$ used in the messages RFB-4 and RFB-5

[4]In the Key Continuity Model of key management, users are prompted to approve K_S manually when connecting a new server S. Implementations should not present K_S to the user or store it until the peer proves possession of K_S^{-1}.

are calculated as follows:

$$\begin{aligned}
Digest_{SV} &= \mathrm{PRF}_{\text{“}\texttt{RfbServerDigest}\text{”}\,\|\,0}(Master, Cookie_V \,\|\, Cookie_S \,\|\, Context),\\
Digest_{VS} &= \mathrm{PRF}_{\text{“}\texttt{RfbClientDigest}\text{”}\,\|\,0}(Master, Cookie_V \,\|\, Cookie_S \,\|\, Context),\\
SymKey_{SV} &= \mathrm{PRF}_{\text{“}\texttt{RfbServerKeyingMaterial}\text{”}\,\|\,0}(Master, Cookie_V \,\|\, Cookie_S \,\|\, Session),\\
SymKey_{VS} &= \mathrm{PRF}_{\text{“}\texttt{RfbClientKeyingMaterial}\text{”}\,\|\,0}(Master, Cookie_V \,\|\, Cookie_S \,\|\, Session),
\end{aligned}$$

where, using the Diffie–Hellman procedure $\mathrm{DH}(a,b) = a^b$,

$$\begin{aligned}
Context &= RFB_V \,\|\, CipherList \,\|\, RFB_S \,\|\, Cipher \,\|\, J_V \,\|\, J_S \,\|\, K_V \,\|\, K_S,\\
Z &= \mathrm{DH}(J_S, J_V^{-1}) = \mathrm{DH}(J_V, J_S^{-1}),\\
Master &= \mathrm{PRF}_{\text{“}\texttt{RfbMasterSecret}\text{”}}(Z \,\|\, N_V \,\|\, N_S, Cookie_V \,\|\, Cookie_S),\\
Session &= \mathrm{PRF}_{\text{“}\texttt{RfbSessionId}\text{”}}(Master, Cookie_V \,\|\, Cookie_S).
\end{aligned}$$

4.3 RA4 algorithm choices

The RSA operations performed in the RA4-2 and RA4-3 messages use OAEP padding, an all-or-nothing padding scheme which can be implemented to leak no information about decrypted bytes to attackers unless the entire padding is correct. (The hash for the mask generation function is SHA-256 for RA4-128 and RA4ne-128, and SHA-512 for RA4-256 and RA4b-256, with a matching salt length enforced.) A minimum RSA key size is enforced by the protocol, of 1024 bits for RA4-128 and RA4ne-128, and 2048 bits for RA4-256 and RA4b-256, and although all RealVNC products currently use 2048-bit keys by default, this may be configured by customers.

The symmetric encryption used in the RFB-4 and RFB-5 messages uses AES in GCM mode, with the MAC provided by the GCM authentication tag. RA4-128 and RA4ne-128 use AES-128, and RA4-256 and RA4b-256 use AES-256. Encrypted packets in RA4 have a length header which is passed as Additional Authenticated Data to the GCM state, followed by that many bytes of AES ciphertext, followed by a full 16-byte GCM authentication tag. The first 16 or 32 bytes of $SymKey_{SV}$ and $SymKey_{VS}$ are used as the AES key, which is used for all messages in that direction; the next 12 bytes are the GCM IV for the first packet, which is incremented for each subsequent packet, preventing replay attacks using a previous MAC.

The PRF used is the HKDF key derivation function (RFC 5869), a standardised building block for application protocols which uses HMAC in counter-mode to stretch source material into uniform bytes.[5] The HMAC hash function is SHA-256 for RA4-128 and RA4ne-128, and SHA-512 for RA4-256 and RA4b-256.

[5] HKDF is defined as follows, expanding its input to any required length:

$$\begin{aligned}
\mathrm{HKDF}(Salt, SourceMaterial, Context) &= \mathrm{Expand}(0) \,\|\, \ldots \,\|\, \mathrm{Expand}(n)\\
\mathrm{PRF}_P(SourceMaterial, Context) &= \mathrm{HKDF}(P, SourceMaterial, Context)
\end{aligned}$$

where Expand(i) is computed by the following process:

$$\begin{aligned}
ExtractedKey &= \mathrm{HMAC}(Salt, SourceMaterial)\\
\mathrm{Expand}(i) &= \begin{cases} \mathrm{HMAC}(ExtractedKey, Context \,\|\, 0) & \text{if } i = 0\\ \mathrm{HMAC}(ExtractedKey, \mathrm{Expand}(i-1) \,\|\, Context \,\|\, (i-1)) & \text{if } 1 \le i < n \end{cases}
\end{aligned}$$

The Diffie–Hellman exchange for RA4 uses the elliptic curve Diffie–Hellman algorithm (ECDH) over `curve25519` (RFC 7748 section 4.1) for RA4-128, RA4ne-128, and RA4-256, providing Perfect Forward Secrecy at the 128-bit security level; for RA4b-256 `curve448` (RFC 7748 section 4.2) is used, providing Perfect Forward Secrecy at the 224-bit level.

4.4 Desired properties of the RA4 flow

We aim to establish that V believes the following statements:

V.1 S believes that RFB-1 had contents RFB_S (V wishes to be sure that S did not in fact wish to offer a later, more secure RFB version)

V.2 S believes that RFB-2 had contents $RFB_V, CipherList$ (V wishes to be sure that S chose from the ciphers V wanted to offer)

V.3 S believes that RFB-3 had contents $Cipher$ and that RA4-1 had contents K_S (V wishes to be sure that S intended to choose RA4 and that S did in fact offer K_S)[6]

V.4 S believes that RA4-2 had contents K_V (V wishes to be sure that an attacker has not substituted his own identity for K_V)

V.5 The symmetric key $SymKey_{SV}$ is being used from S to V (that is, V wishes to be sure that S has sent a message which attests to this key, and that the verification message is fresh)

V.6 The symmetric keys $SymKey_{SV}$ and $SymKey_{VS}$ are secret between V and S

We aim to establish that S believes the following statements:

S.1 V believes that RFB-1 had contents RFB_S (S wishes to be sure that V chose from the RFB version which S wanted to offer)

S.2 V believes that RFB-2 had contents $RFB_V, CipherList$ (S wishes to be sure that V is not in fact sending RFB messages with subtly different semantics from what it expects, using a different RFB version)

S.3 V believes that RFB-3 had contents $Cipher$ and that RA4-1 had contents K_S (S wishes to be sure that V is aware S chose RA4, and that V saw the key S intended to offer)[7]

S.4 V knows K_V^{-1}, which is weaker than the assurance V.4 that V will gain about S. This difference is because we assumed K_V to be ephemeral, and K_S to be well-known, so S isn't allowed to care about who V is at this stage in the protocol! Instead, a knowledge that V possesses K_V^{-1} is adequate to establish the final property below that the AES keys are a shared secret that can be used for communication. The viewer is anonymous to the server until the later *authentication* stage of the protocol.

S.5 The symmetric key $SymKey_{VS}$ is being used from V to S

S.6 The symmetric keys $SymKey_{SV}$ and $SymKey_{VS}$ are secret between V and S

[6] These properties are of relatively minor importance; an attack relying on corrupting these messages would be quite obscure in practice — assuming that K_S is checked against a well known value.

[7] As for V.3, this is of minor importance in practice.

4.5 Derivation of the properties of the RA4 flow

The properties V.1–V.4 and S.1–S.4 are established as follows: all of these messages have their contents included in *Context*, which is part of the input to both $Digest_{SV}$ and $Digest_{VS}$. Therefore, it is sufficient to establish that S sent $Digest_{SV}$ and V sent $Digest_{VS}$ and that those messages are known to be fresh in order to conclude that V believes V.1–V.4 and S believes S.1–S.4.

We assume under the security policy that only S possesses K_S^{-1}, and we show that S does not provide a decryption oracle for its key. The only private-key operation exposed by the protocol is the decryption operation on $\{N_V\}_{K_S}$ required to compute *Master*, but after decryption the nonce N_V is combined using the PRF with $Cookie_S$, a non-repeating server-provided value, and the PRF is preimage resistant, so the decrypted value N_V is not made available to attackers and we see that S does not expose a decryption oracle for K_S^{-1} in the RFB-4 message. Under the security policy we consider S and V do not send messages outside of the protocol, allowing us to conclude that there is no decryption oracle available for K_S^{-1} and that V does not send N_V over the network either, and therefore only V and S know N_V. The same holds for N_S by the same argument for V with K_V^{-1}.

The fact that S sent $Digest_{SV}$ and V sent $Digest_{VS}$ is seen from the way N_V and N_S are stirred in to the digests using the second-preimage-resistant PRF, so that computing the correct value of the digests is no easier for the attacker than finding the values of the nonces. We know that only V and S know the nonces, therefore, V knows that S sent $Digest_{SV}$ and S that V sent $Digest_{VS}$.

Finally, we observe that the digests are known to be fresh, since the cookies $Cookie_V$ and $Cookie_S$ act as challenges, by being hashed into the final digests. As long as V does not reuse nonces, V can be certain that a replay attack is not in effect (by the second-preimage-resistance of the PRF, which states that it is vanishingly likely that two randomly-chosen cookies will produce the same digest), and similarly for S. We have a very desirable property as well: V does not rely on S providing a random cookie or nonce, nor does S rely on V's cookie and nonce generation. If S repeats a cookie, its assurances are diminished, but V's are unaffected, and since both nonces are required for calculating the digests, if one of the nonces (but not both) is known to an attacker he gains no advantage in spoofing the digests. Therefore, neither party need rely on the other at any stage in the protocol to generate random values securely; if they themselves are confident that their own generated nonces are not known to an attacker, their own assurances are undiminished.

We now consider the properties V.5–V.6 and S.5–S.6. The fact that the symmetric keys are secret follows from the previous discussion, in which the output of the PRF required knowledge of the input, which is derived from the secret values J_V^{-1} and J_S^{-1}. The beliefs that V and S are using the symmetric keys to communicate (V.5 and S.5) follow from receiving the correct verification digest, and knowing that these are fresh by virtue of the use of fresh cookies in each exchange.

The last property to demonstrate is the Perfect Forward Secrecy property. Suppose that the attacker gains access to K_V^{-1} and K_S^{-1} after the peers have stopped sending and reading messages. At that time, only passive attacks are possible, in which the attacker decrypts previous protocol data. All secret values in the protocol are protected using the symmetric keys $SymKey_{VS}$ and $SymKey_{SV}$ which depend on Z, but Z is determined by the ephemeral Diffie–Hellman keys rather than K_V^{-1} and K_S^{-1} so no additional information is gained by the attacker.

5 Analysis of RA4-PSK

5.1 Security policy for VNC Cloud and RA4-PSK

The policy under which we analyse RA4-PSK extends the policy used for RA4 above. Although the RFB connection is preceded by a VNC Cloud handshake, the communication between the viewer and server proceeds as before, under the same assumptions.

We extend the RA4 security policy in two ways, firstly by exchanging the RSA keys via VNC Cloud, and secondly by exchanging a Pre-Shared Key (nonce).

Exchanging the RSA keys via VNC Cloud relaxes the requirement in section 4.1 that the viewer must already know the server's public key K_S: if the viewer already knows the server's key, it may use that information as an additional check, but if the connection is being made for the first time then the secure exchange of keys via the VNC Cloud connection broker is sufficient. In the same way, the viewer's key K_V is no longer considered to be ephemeral, but is known to the server and can be checked.

The assumption that the VNC Cloud connection broker is secure is a strong one, and users of VNC Cloud may have differing levels of trust in VNC Cloud. Although communication with VNC Cloud is performed over encrypted TLS connections, and VNC Cloud is hosted in secure data-centres, some users may have concerns over whether an intruder has subverted the cloud servers, or whether RealVNC itself can be trusted. In these circumstances, the attack scenario is that the connection broker service may substitute the server's actual RSA key with an impersonator's key K_I, leading the viewer to believe that K_I is the S's key. The attacker may then impersonate S using K_I^{-1} without needing to obtain K_S^{-1}.

To describe the levels of trust placed in VNC Cloud, we refine the security policy to describe three types of user:

1. Most users trust VNC Cloud. There is no commercial incentive for RealVNC itself to impersonate their customers' servers, and they accept the level of security used to control access to the VNC Cloud data-centres by third parties. In this case, the policy is straightforward and connection brokering service is considered to establish the keys K_V and K_S securely.

2. Users who provide their own remote-access system based on VNC Cloud may wish to add their own exchange of RSA keys, in addition to the RSA key exchange via VNC Cloud. In this scenario, before the RFB connection begins the viewer and server will additionally check the RSA keys against another web-service.[8] If both sources of authority match, then the keys K_S and K_V are considered to be established securely.

3. A very small number of users may not wish to trust any third-party authority at all. In this case, they may configure their VNC Viewer not to trust VNC Cloud, and the user is prompted to manually approve all server keys. The security policy for this use-case is the same as the policy considered above for direct connections using RA4.

Secondly, we extend the RA4 security policy with an exchange of Pre-Shared Keys. During the connection broker handshake, nonces PSK_V and PSK_S are sent. VNC Cloud

[8]The VNC Developer SDK provides callbacks in order to enable this level of authentication. The SDK user is assumed to have their own web-service (in order to exchange VNC Cloud addresses), so it is possible to exchange the RSA keys along with the Cloud addresses.

guarantees that the nonces will be kept secret, and only sent once to the peer. The security policy for RA4-PSK leverages this, by requiring that the peer prove possession of both nonces. If this is the case, then the peer's identity must match the identify of the target Cloud address (since VNC Cloud authenticates all uses of Cloud addresses). Furthermore, if a Cloud address is conducting two simultaneous connections, each Cloud handshakes is bound to its corresponding RA4-PSK connection, so this policy prevents redirection attacks, where an attacker attempts to blindly redirect traffic for one connection to another. The policy dictates that PSK_V and PSK_S are only used by the viewer and server respectively in a single RFB connection and in a single VNC Cloud handshake.

5.2 RA4-PSK messages

Before the RFB connection begins, the viewer and server perform a VNC Cloud handshake via the connection broker C, using their VNC Cloud addresses $Addr_V$ and $Addr_S$. In the discussion that follows, we only consider the cryptographic material, but they also exchange IP addresses and other information required to set up the subsequent RFB connection. These messages are sent over an authenticated, secure channel to VNC Cloud.

$$V \rightarrow C : K_V, PSK_V, Addr_S \tag{CLOUD-1}$$
$$C \rightarrow S : K_V, PSK_V \tag{CLOUD-2}$$
$$S \rightarrow C : K_S, PSK_S, Addr_V \tag{CLOUD-3}$$
$$C \rightarrow V : K_S, PSK_S \tag{CLOUD-4}$$

The RFB messages for RA4-PSK are identical to those shown for RA4 in section 4.2, shown here again as a reminder:

$$S \rightarrow V : J_S, K_S \tag{RA4-PSK-1}$$
$$V \rightarrow S : J_V, K_V, \{N_V\}_{K_S} \tag{RA4-PSK-2}$$
$$S \rightarrow V : \{N_S\}_{K_V} \tag{RA4-PSK-3}$$

The only difference concerns the calculation of the shared secret, which is derived as follows using the addition of the Pre-Shared Keys:

$$Z = \mathrm{DH}(J_S, J_V^{-1}) = \mathrm{DH}(J_V, J_S^{-1}),$$
$$Master = \mathrm{PRF}_{\text{“}\texttt{RfbMasterSecret}\text{”}}(Z \,\|\, N_V \,\|\, N_S \,\|\, PSK_V \,\|\, PSK_S, Cookie_V \,\|\, Cookie_S).$$

5.3 RA4-PSK algorithm choices

The same algorithms are used as for RA4; in particular, RA4-PSK-128 uses the same algorithms as RA4-128 with the addition of the Pre-Shared Key; RA4-PSK-256 uses the same algorithms as RA4-256, and RA4b-PSK-256 the same as RA4b-256.

5.4 Properties of the RA4-PSK flow

The important properties of the RA4-PSK flow are largely identical to those of the RA4 flow. We note that the requirement S.4 from section 4.4 must be slightly amended for RA4-PSK to take into account the fact that K_V is now known to S rather than being ephemeral. The requirement that V proves possession of K_V^{-1} and that S proves possession of K_S^{-1} follows from the previous analysis.

The only new property to demonstrate is that the peers prove possession of both of PSK_V and PSK_S. This however is clear from the construction of *Master* using the second-preimage-resistant PRF to show that *Master* cannot be computed without the Pre-Shared Keys, and that *Master* is required to compute the verification digests.

Finally, using the policy's requirement that the Pre-Shared Keys are not reused in more than one handshake or RFB connection, we know that the RFB peer must correspond to the same peer as in the VNC Cloud handshake.

6 Analysis of RA4-PSK-Only

6.1 Security policy for RA4-PSK-Only

For the VNC Connect "Instant Support" product, the additional cipher suites RA4-PSK-Only are offered. These are broadly similar to RA4-PSK, but without the RSA handshake.

This is motivated by the following security model: a remote user contacts a customer support technician (perhaps over the telephone), and requests assistance from the technician (the customer support technician is not a RealVNC employee, but a RealVNC customer who uses RealVNC's product to provide a service in turn to their users).[9] In this case, we model the two endpoints as having no prior knowledge of each other, apart from an out-of-band connection (the telephone number), and as having not exchanged cryptographic keys.

In the Instant Support product, the technician directs the user to a URL from which the user downloads and runs the Instant Support server application. The technician initiates the connection using their viewer, and reads out a One-Time Code to the remote user. Once the user has entered this code into the Instant Support server, a connection between the support technician's viewer application and the server is brokered via VNC Cloud.

As before, we are not concerned in this document with the details of the VNC Cloud exchange, and only consider the RFB connection that results.

Since the remote user and support technician have not shared any long-term keys, the RA4-PSK-Only cipher suites differ from RA4-PSK by not using a pair of RSA identity keys. The One-Time Code entered on the server end locks down the server application, such that it will only accept a connection from the corresponding technician's viewer,

[9]We refer to the technician as the "trusted technician" on the basis that the remote user ought to have a reason to trust the person they are communicating with; for example, the technician may be an employee of a company which the user trusts. In the flow just outlined, the trust relationship is established when the *user initiates the telephone call*, which provides protection against phishing attacks.

In this document, we focus on the cryptographic measures of the RFB protocol, and do not specifically consider application-level concerns such as phishing protection. However, the overall security model of the RealVNC products certainly does take these considerations into account, and RealVNC takes care throughout the product design process to prevent where possible the use of the Instant Support product for phishing attacks. RealVNC does not encourage customer support flows which train users to become accustomed to dangerous practices (such as trusting incoming telephone calls). Before the incoming connection is permitted, the user is prompted to confirm, "I trust the technician about to connect", and the capabilities that will be granted to the technician are described.

Finally, the Instant Support server is designed to keep the user in charge, able to disconnect the remote technician at any time.

as identified solely by the VNC Cloud negotiation service. The user must trust the VNC Cloud servers to be faithfully relaying any secrets exchanged, and has no external means to verify that the incoming viewer connection does belong to the trusted technician.

The security policy therefore operates under the assumption that the exchanged keys PSK_V and PSK_S are securely associated with the One-Time Code when exchanged via VNC Cloud services. The same restrictions on the re-use of the Pre-Shared Keys apply as in section 5.1. The policy's primary objective is to ensure that no untrusted viewer is able to connect to the Instant Support server, and secondarily that the technician's viewer shall only receive connections from users who know the One-Time Code.

6.2 RA4-PSK-Only messages

Before the RFB connection begins, the support technician's viewer requests a One-Time Code OTC from the VNC Cloud services C, and the support technician relays this out-of-band (perhaps via telephone) to the remote user, who finally provides it to the Instant Support server. Then, the connection is initiated by further requests to the Cloud services. We only consider here the exchange of the cryptographic material, although the actual messages also include IP addresses and other information required to set up the subsequent RFB connection.

$$V \to C : PSK_V \tag{CLOUD-SUPPORT-1}$$
$$C \to V : OTC \tag{CLOUD-SUPPORT-2}$$
$$V \to S : OTC \tag{exchanged out-of-band}$$
$$S \to C : OTC, PSK_S \tag{CLOUD-SUPPORT-4}$$
$$C \to S : PSK_V \tag{CLOUD-SUPPORT-5}$$
$$C \to V : PSK_S \tag{CLOUD-SUPPORT-6}$$

The RFB messages for RA4-PSK-Only differ from those shown for RA4-PSK in section 5.2, by including only the ephemeral Diffie–Hellman keys:

$$S \to V : J_S \tag{RA4-PSK-ONLY-1}$$
$$V \to S : J_V \tag{RA4-PSK-ONLY-2}$$

The calculation of the shared secret is also updated as follows:

$$Z = \mathrm{DH}(J_S, J_V^{-1}) = \mathrm{DH}(J_V, J_S^{-1}),$$
$$Master = \mathrm{PRF}_{\text{“}\texttt{RfbMasterSecret}\text{”}}(Z \,\|\, PSK_V \,\|\, PSK_S, Cookie_V \,\|\, Cookie_S).$$

6.3 RA4-PSK-Only algorithm choices

The same algorithms are used as for RA4-PSK; in particular, RA4-PSK-Only-128 uses the same algorithms as RA4-PSK-128 with the exception of the removed RSA nonce exchange; RA4-PSK-Only-256 uses the same algorithms as RA4-PSK-256, and RA4b-PSK-Only-256 the same as RA4b-PSK-256.

6.4 Properties of the RA4-PSK-Only flow

The key properties of RA4 are unchanged from before, ensuring that the encryption keys negotiated allow confidential and authenticated exchange of data.

The properties specific to the Instant Support flow revolve around the changes to the identity verification procedure. Under the assumption that the code OTC is securely associated with a unique technician by the Cloud services, then it follows that S may be certain that only V, S (and C) have knowledge of each of PSK_V and PSK_S. Further, the key PSK_V was obtained from C through its association with OTC, hence S may conclude that V has knowledge of OTC. This is the primary desired security property, that the server is able to verify that only the authorised technician is permitted to connect, as identified by the One-Time Code entered when the server application starts.

In addition, V may be certain that the RFB server S has knowledge of PSK_V by virtue of the construction of $Master$, and then, from the association with the code OTC enforced by the Cloud services, it follows that S must have obtained this through knowledge of OTC. This is the secondary security property, that V knows that the server end of the RFB connection has access to the One-Time Code exchanged over the out-of-band channel.

Note that we implicitly required above the additional assumption that C discards and does not make use of the Pre-Shared Keys exchanged via its Cloud services, in order for S to conclude that the incoming connection originates from the trusted technician V. This differs from ordinary Cloud connections covered in section 5, in which it was not essential to rely on the trustworthiness of RealVNC's Cloud services C, since their faithful operation could be verified using the long-term keys K_V and K_S on the viewer end, and, on the server end ultimate authentication decisions reside solely in the client software. The strengthened assumption on RealVNC's services acting as a trusted central authority is considered unavoidable for the Instant Support use-case, which connects two endpoints previously unknown to each other.

7 Analysis of other RFB 5.0 properties

7.1 Changing encryption level

RFB 5.0 includes messages designed to allow a cipher to offer an unencrypted level with integrity only, stemming from a business requirement to additionally support existing secure environments where encryption is prohibitively expensive.

However, if the session is not encrypted, we still must not send passwords or other authentication information in the clear. So, the protocol must initially wrap the traffic in encrypted packets, then switch to unencrypted packets after the authentication phase. RA4ne-128 provides two encryption levels: "level 1" is AES-128, "level 2" is plaintext with an HMAC appended (the HMAC includes a packet counter as well as the plaintext to prevent replays).

For RA4ne-128, it is a protocol error to switch to level 2 before the authentication phase has ended. The other RFB 5.0 cipher suites do not use multiple levels.

In order to implement these requirements, RFB 5.0 provides the RFB-LEVEL message which resets the encryption keys and cipher level in one direction:

$$V \to S \text{ or } S \to V : \{NewLevel\}_{SymKey_n}, \{Digest_{n+1}\}_{SymKey_{n+1}} \tag{RFB-LEVEL}$$

The first part of the message is encrypted using the old symmetric key, just like all RFB traffic after the cipher is established, and it contains the new cipher level *NewLevel* (which may be equal to the current level). The second half of the message contains a confirmation digest, encrypted using the new encryption level and with a new key. Note that $Digest_{n+1}$ is sent as authenticated plaintext when switching to an unencrypted level. The new digest and key are calculated as follows:

$$Digest_{n+1} = \begin{cases} \mathrm{PRF}_{\text{“}\texttt{RfbServerDigest}\text{”}\,\|\,n}(Master, Cookie_V \,\|\, Cookie_S) & \text{for } S \to V \\ \mathrm{PRF}_{\text{“}\texttt{RfbClientDigest}\text{”}\,\|\,n}(Master, Cookie_V \,\|\, Cookie_S) & \text{for } V \to S \end{cases}$$

$$SymKey_{n+1} = \begin{cases} \mathrm{PRF}_{\text{“}\texttt{RfbServerKeyingMaterial}\text{”}\,\|\,n}(Master, Cookie_V \,\|\, Cookie_S \,\|\, Session) \\ \hfill \text{for } S \to V \\ \mathrm{PRF}_{\text{“}\texttt{RfbClientKeyingMaterial}\text{”}\,\|\,n}(Master, Cookie_V \,\|\, Cookie_S \,\|\, Session) \\ \hfill \text{for } V \to S \end{cases}$$

Each time the level is changed, a fresh digest and keying material are generated using a different parametrization of the PRF, ensuring that the new materials are independent of the old, and HMAC and AES keys are never re-used. The digest serves as an error checking mechanism.

Because RA4-128 and RA4-256 have only one level, downgrade attacks are not possible, nor are renegotiation attacks: the RFB cipher suite is picked once only in the RFB-3 message and cannot be changed. The level switching is purely a way for RA4ne-128 to select between different options within the already-negotiated cipher suite.

The formal requirements for the new keying material $SymKey_{n+1}$ are that it is secret between V and S, and that the peer believes the key to be in use. Secrecy is assured for the n-th symmetric key identically as for the first key, namely that the output of the PRF cannot be known to an attacker since its input *Master* is still secret. The peer's belief that the key is in use could be deduced without the presence of the digest, since the peer has previously attested to the inputs *Master*, *Session*,[10] and the cookies, but the fact that a new digest is sent provides a robust double-check that the new keying material has been correctly brought into use.

7.2 Re-keying

The final RFB cryptographic feature we analyse allows the keying material to be re-generated, and has two motivations: a) We have a multi-process architecture including sandboxed processes in various products. To allow implementers to isolate privileges effectively, we provide a way for the master key to be incremented irreversibly so that a connection can be handed over to another process, without that process being able to decrypt traffic sent or received before the transfer.[11] b) Secondly, RFB contains many

[10]Stirring *Session* in to the symmetric keys provides added robustness. The master key can be iterated during the running of the protocol (see section 7.2 below), which may lead to a degeneration of the entropy left in the symmetric keys after many rekeying operations. In practice, iterated application of HMAC-SHA256 is believed to have cycles of length up to 2^{128}, and there are no known degenerative cycles.

[11]A product could for example use Capsicum sandboxing and privilege separation to isolate its components. The component providing password management could re-key the connection before handing it over to a process in another sandbox, to prevent lower-privilege components from having access to data sent by higher-privilege components. This design is used by the `RootSecurity` feature in VNC Server.

packets with plaintext easily guessable from traffic analysis, and encrypting many gigabytes of data with known plaintext chunks using AES is considered inadvisable, so re-keying may be useful in future to guard against attacks on long-running RFB sessions.[12]

The re-key messages can be initiated by either peer, at any time, except when a re-key is already in progress. The messages sent make up a three-way handshake (an exceptional four-way handshake is also possible[13]):

$$A \rightarrow B : \{Cookie_A^{n+1}\}_{SymKey_{AB}^n} \tag{RFB-KEY-1}$$

$$B \rightarrow A : \{Cookie_B^{n+1}\}_{SymKey_{AB}^n}, \{Digest_{BA}^{n+1}\}_{SymKey_{BA}^{n+1}} \tag{RFB-KEY-2}$$

$$A \rightarrow B : \{Digest_{AB}^{n+1}\}_{SymKey_{AB}^{n+1}} \tag{RFB-KEY-3}$$

The encryption method is not changed by a re-key; the cipher's level remains the same. The new digest and key are calculated as follows:

$$Master_{n+1} = \mathrm{PRF}_{\text{“}\texttt{RfbMasterSecret}\text{”}}(Master_n, Cookie_V^{n+1} \,\|\, Cookie_S^{n+1} \,\|\, Session)$$

$$Digest_{PQ}^{n+1} = \begin{cases} \mathrm{PRF}_{\text{“}\texttt{RfbServerDigest}\text{”}\|0}(Master_{n+1}, Cookie_Q^{n+1} \,\|\, Cookie_P^{n+1}) \\ \qquad\qquad\qquad \text{for } (P,Q) = (S,V) \\ \mathrm{PRF}_{\text{“}\texttt{RfbClientDigest}\text{”}\|0}(Master_{n+1}, Cookie_P^{n+1} \,\|\, Cookie_Q^{n+1}) \\ \qquad\qquad\qquad \text{for } (P,Q) = (V,S) \end{cases}$$

$$SymKey_{PQ}^{n+1} = \begin{cases} \mathrm{PRF}_{\text{“}\texttt{RfbServerKeyingMaterial}\text{”}\|0}(Master_{n+1}, Cookie_Q^{n+1} \,\|\, Cookie_P^{n+1} \,\|\, Session) \\ \qquad\qquad\qquad \text{for } (P,Q) = (S,V) \\ \mathrm{PRF}_{\text{“}\texttt{RfbClientKeyingMaterial}\text{”}\|0}(Master_{n+1}, Cookie_P^{n+1} \,\|\, Cookie_Q^{n+1} \,\|\, Session) \\ \qquad\qquad\qquad \text{for } (P,Q) = (V,S) \end{cases}$$

(Note that the “level counter” in the PRF parameter for the digests and keys is reset to zero.)

We desire to show that the new symmetric keys are a secret between the server and viewer, and that the keys are in use. The reasoning proceeds in the same way as in section 7.1 and is not repeated again.

[12] Due to birthday attacks AES becomes weak at around 2^{68} bytes, not accounting for leaks provided by known plaintext blocks. Historically, protocols have included frequent re-keying due to problems with DES, and attacks on some block cipher modes require re-keying after comparatively small amounts of data, such as 2^{48} blocks for OCB (RFC 7253 section 5). Although such a weakness has not yet been published for AES in GCM mode, it is prudent to provide re-keying as part of a protocol.

[13] If both peers initiate a re-key simultaneously, the following messages are sent, with RFB-KEY-LUCKY-1 transmitted simultaneously with RFB-KEY-LUCKY-2:

$$A \rightarrow B : \{Cookie_A^{n+1}\}_{SymKey_{AB}^n} \tag{RFB-KEY-LUCKY-1}$$

$$B \rightarrow A : \{Cookie_B^{n+1}\}_{SymKey_{BA}^n} \tag{RFB-KEY-LUCKY-2}$$

$$A \rightarrow B : \{Digest_{AB}^{n+1}\}_{SymKey_{AB}^{n+1}} \tag{RFB-KEY-LUCKY-3}$$

$$B \rightarrow A : \{Digest_{BA}^{n+1}\}_{SymKey_{BA}^{n+1}} \tag{RFB-KEY-LUCKY-4}$$

The general scheme is not affected by the messages arriving in this way.

The new property of the exchange to note is that $Master_n$ is not recoverable from $Master_{n+1}$. If a party solely knows the new symmetric keys and new master secret, it is not possible to decrypt any messages exchanged under the old keys, since the PRF uses a one-way hash function to prevent recovery of the old keying material.