Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 10, 2016

C. Jennings
Cisco Systems

May 9, 2016

Using WebRTC Identity with STIR
draft-jennings-stir-rtcweb-identity-00

Abstract

This draft outlines the approach to use STIR Passport tokens as the Identity assertions in WebRTC. It outlines some proposed changes to improve identity interoperability for different calling systems.

This is a very rough first draft purely to have something to discuss.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on November 10, 2016.

Table of Contents

1. Introduction

This draft is purely to help track the discussion about WebRTC and STIR identity and provide a place to send pull requests. It is not meant to become an RFC - the relevant information will flow into other specifications once we come to some consensus on what needs to be done.

It assumes the reader if familiar with [I-D.ietf-rtcweb-security-arch], [I-D.ietf-stir-certificates], [I-D.ietf-stir-passport], and [I-D.ietf-stir-rfc4474bis].

2. Information Flow

This section steps through the logical information flow to use STIR Identity in WebRTC based on what we have in the specifications today. Later sections propose some changes to current specifications.

2.1. Actors

Cullen, with a phone number of +1 408 555 1212 is using skype.com to call Jon at "sip:jon@example.org". Both people use a WebRTC softphone. Cullen uses an identity provider of att.com. Skype could be the identity provider for this use case but it's easier to understand the flow if you look at it as a separate provider.

Before the call ever starts, Cullen's browser fetches the webpage from skype.com and logs on to that website. Cullen then goes to call Jon. The skype JS notes that the Cullen's identity provider is att.com and the JavaScript for the skype web page does a

```
pc.setIdentityProvider("att.com","passport",
                       JSON.stringify( {"otn": "14085551212",
                                        "duri": "sip:jon@example.org"}));
```

And start the setup of media to the other side which will eventually generate an SDP offer. The browser will load the IDP JS from IdP at

https://att.com/.well-known/idp-proxy/passport

(see section 5.6.5 of [I-D.ietf-rtcweb-security-arch])

The browser will then call the IDP javascript code to generate a WebRTC identity assertion. It will pass to the generating function the hint information provided in the setIdentityProvider of

```
{ "otn": "14084219990",
   "duri": "sip:jon@example.org" }
```

The browser also passes the DTLS fingerprints and hash algorithm used to generate them to the JS for the IdP in an array like

```
[
   {
      "algorithm":"sha-256",
      "digest":"23:E8:6E:28:3F:D1:CF:17:DE:88:0F:EE:5A:AB:
          AD:03:2D:77:FB:05:17:BA:61:12:1B:37:D4:19:4F:38:DF:EC"
   }
]
```

All of this information can be sent to the idp.com along with the whatever login flow att.com wants to use to have Cullen authenticated to att.com

At this point, the IdP has all the information to create the Passport Token.

The DTLS-SRTP fingerprints are reformatted to be in the form

```
"mky": "sha-256 23:E8:6E:28:3F:D1:CF:17:DE:88:0F:EE:5A:AB:
             AD:03:2D:77:FB:05:17:BA:61:12:1B:37:D4:19:4F:38:DF:EC"
```

The passport header is set to

```
            {
               "typ":"passport",
               "alg":"RS256",
               "x5u":"https://cert.att.org/passport.crt"
            }
```

The current time is used for the iat field and a passport payload is formed. Note the payload also gets called the claims. For example, the payload could look like

```
{
   "iat":"14443208345",
   "otn":"14084219990",
   "duri":"sip:jon@example.org",
   "mky":"sha-256 23:E8:6E:28:3F:D1:CF:17:DE:88:0F:EE:5A:AB:
             AD:03:2D:77:FB:05:17:BA:61:12:1B:37:D4:19:4F:38:DF:EC"
}
```

This is canonicalized and signed with a certificate valid to sign 14084219990. The URL to fetch the certificate is found in the x5u field of the Passport Header.

The canonical headers base64 is

```
eyJ0eXAiOiJwYXNzcG9ydCIsImFsZyI6IlJTMjU2IiwieDV1IjoiaHR0cHM6Ly9jZXJ0
LmV4YW1wbGUub3JnL3Bhc3Nwb3J0LmNydCJ9
```

The canonical payload in base64 is

```
eyJpYXQiOiIxNDQ0MzIwODM0NSIsIm90biI6IjE0MDg0MjE5OTkwIiwiZHVyaSI6InNp
cDpqb25AZXhhbXBsZS5vcmciLCJta3kiOiJzaGEtMjU2IDIzOkU4OjZFOjI4OjNGOkQx
OkNGOjE3OkRFOjg4OjBGOkVFOjVBOkFCOkFEOjAzOjJEOjc3OkZCOjA1OjE3OkJBOjYx
OjEyOjFCOjM3OkQ0OjE5OjRGOjM4OkRGOkVDIn0
```

The base64 headers and payload are concatenated with a period between them and the SHA 256 of the result is computed which, represented in hex, is

```
  20ac9439ae10df640b39e177d4ec432fe590b2d95f4075957816b5929ca1acb0
```

This is signed using RSASSA-PKCS1-v1_5 to get a base64 encoded signature of

FMgDzm0ZLV3w-6yOSCm3ZorSwEednS38XtF2u29NhRQ5zh8Swh0uSGHASXFwUIQ7DXvG
4pKD1rbBLFYT3_ReDn6OJ36k0S3oNvwMD0FyrRYeji-pAjoh76Q-jsfJsOq-Pi35Fba8
ZYRvG_EgLE0FYXuM2nWm85RUKVSnkx5MBpOxuUxRyTZYJpaQjElVpv9MJKfd_HyuF66q
98tcqeDb8dlrjtPNnQQfOf6AyLTlRwJ64t7iTouGD_R85Lv4Hu4mwu2ZiOL_1jo5svoY
-KegM1KJh-M_ux4J52NoVeeLLgKg585qUHLrln6aEjevNQLBUcbdf_DKZ1hyj0oFR9lx
rw

The above example was done with and RSA key size of 2048 bits.

The headers, payload, and signature are concatenated with pereidos between then to form a Passport Token of the form (but with no line breaks)

eyJ0eXAiOiJwYXNzcG9ydCIsImFsZyI6IlJTMjU2IiwieDV1IjoiaHR0cHM6Ly9jZXJ0
LmV4YW1wbGUub3JnL3Bhc3Nwb3J0LmNydCJ9.
eyJpYXQiOiIxNDQ0MzIwODM0NSIsIm90biI6IjE0MDg0MjE5OTkwIiwiZHVyaSI6InNp
cDpqb25AZXhhbXBsZS5vcmciLCJta3kiOiJzaGEtMjU2IDIzOkU4OjZFOjI4OjNGOkQx
OkNGOjE3OkRFOjg4OjBGOkVFOjVBOkFCOkFEOjAzOjJEOjc3OkZCOjA1OjE3OkJBOjYx
OjEyOjFCOjM3OkQ0OjE5OjRGOjM4OkRGOkVDIn0.
FMgDzm0ZLV3w-6yOSCm3ZorSwEednS38XtF2u29NhRQ5zh8Swh0uSGHASXFwUIQ7DXvG
4pKD1rbBLFYT3_ReDn6OJ36k0S3oNvwMD0FyrRYeji-pAjoh76Q-jsfJsOq-Pi35Fba8
ZYRvG_EgLE0FYXuM2nWm85RUKVSnkx5MBpOxuUxRyTZYJpaQjElVpv9MJKfd_HyuF66q
98tcqeDb8dlrjtPNnQQfOf6AyLTlRwJ64t7iTouGD_R85Lv4Hu4mwu2ZiOL_1jo5svoY
-KegM1KJh-M_ux4J52NoVeeLLgKg585qUHLrln6aEjevNQLBUcbdf_DKZ1hyj0oFR9lx
rw

The IdP returns this to the IdP JS in the browser which forms an object like

```
{
"assertion":
"eyJ0eXAiOiJwYXNzcG9ydCIsImFsZyI6IlJTMjU2IiwieDV1IjoiaHR0cHM6Ly9jZXJ0
LmV4YW1wbGUub3JnL3Bhc3Nwb3J0LmNydCJ9.
eyJpYXQiOiIxNDQ0MzIwODM0NSIsIm90biI6IjE0MDg0MjE5OTkwIiwiZHVyaSI6InNp
cDpqb25AZXhhbXBsZS5vcmciLCJta3kiOiJzaGEtMjU2IDIzOkU4OjZFOjI4OjNGOkQx
OkNGOjE3OkRFOjg4OjBGOkVFOjVBOkFCOkFEOjAzOjJEOjc3OkZCOjA1OjE3OkJBOjYx
OjEyOjFCOjM3OkQ0OjE5OjRGOjM4OkRGOkVDIn0.
FMgDzm0ZLV3w-6yOSCm3ZorSwEednS38XtF2u29NhRQ5zh8Swh0uSGHASXFwUIQ7DXvG
4pKD1rbBLFYT3_ReDn6OJ36k0S3oNvwMD0FyrRYeji-pAjoh76Q-jsfJsOq-Pi35Fba8
ZYRvG_EgLE0FYXuM2nWm85RUKVSnkx5MBpOxuUxRyTZYJpaQjElVpv9MJKfd_HyuF66q
98tcqeDb8dlrjtPNnQQfOf6AyLTlRwJ64t7iTouGD_R85Lv4Hu4mwu2ZiOL_1jo5svoY
-KegM1KJh-M_ux4J52NoVeeLLgKg585qUHLrln6aEjevNQLBUcbdf_DKZ1hyj0oFR9lx
rw",
   "idp":{
      "domain":"https://ks.fluffy.im:10443",
      "protocol":"passport"
   }
}
```


This object is passed back into the browser user agent which base 64 encodes adds it to the SDP in an header like


```
a=identity:eyJhc3NlcnRpb24iOiJleUowZVhBaU9pSndZWE56Y0c5eWRDSXNJbUZzW
nlJNklsSlRNalUySWl3aWVEVjFJam9pYUhSMGNITTZMeTlqWlhKMExtVjRZVzF3YkdVd
WIzSm5MM0JoYzNOd2IzSjBMbU55ZENKOS5leUpwWVhRaU9pSXhORFEwTXpJd09ETTBOU
0lzSW05MGJpSTZJakUwTURnME1qRTVPVGt3SWl3aVpIVnlhU0k2SW5OcGNEcHFiMjVBW
lhoaGJYQnNaUzV2Y21jaUxDSnRhM2tpT2lKemFHRXRNalUySURJek9rVTRPalpGT2pJN
E9qTkdPa1F4T2tOR09qRTNPa1JGT2pnNE9qQkdPa1ZGT2pWQk9rRkNPa0ZFT2pBek9qS
kVPamMzT2taQ09qQTFPakUzT2tKQk9qWXhPakV5T2pGQ09qTTNPa1EwT2pFNU9qUkdPa
k00T2tSR09rVkRJbjAuRk1nRHptMFpMVjN3LTZ5T1NDbTNab3JTd0VlZG5TMzhYdEYyd
TI5TmhSUTV6aDhTd2gwdVNHSEFTWEZ3VUlRN0RYdkcNCjRwS0QxcmJCTEZZVDNfUmVEb
jZPSjM2azBTM29OdndNRDBGeXJSWWVqaS1wQWpvaDc2US1qc2ZKc09xLVBpMzVGYmE4W
llSdkdfRWdMRTBGWVh1TTJuV204NVJVS1ZTbmt4NU1CcE94dVV4UnlUWllKcGFRakVsV
nB2OU1KS2ZkX0h5dUY2NnE5OHRjcWVEYjhkbHJqdFBOblFRZk9mNkF5TFRsUndKNjR0N
2lUb3VHRF9SODVMdjRIdTRtd3UyWmlPTF8xam81c3ZvWVxLZWdNMUtKaE1fdXg0SjUyT
m9WZWVMTGdLZzU4NXFVSExybG42YUVqZXZOUUxCVWNiZGZfREtaMWh5ajBvRlI5bHhyd
yIsImlkcCI6eyJkb21haW4iOiJodHRwczovL2tzLmZsdWZmeS5pbToxMDQ0MyIsInByb
3RvY29sIjoicGFzc3BvcnQifX0=
```


Which adds about 1K bytes to the SIP message.

2.2. Mapping to SIP

A RTCWeb to SIP Gateway can take this assertion, decode it, and extract the Passport Token out of it. It could then insert that into the SIP Invite. It would need to cache the domain and protocol for this flow. Later when a reply came back it, it could take the

Passport token in the Identity and create a new assertion using the cached domain and protocol and insert that. This is really gross and we need to figure out a way where the SDP from SIP is the same as the SDP from WebRTC when using the same passport token.

3. Proposed Changes

3.1. Remove Identity Hint Hack

Passing

```
JSON.stringify({"otn":"14084219990",
                "duri":"sip:jon@example.org"})
```

is sort of a hack. It would be better to pass "14084219990" as the identity hint and have the API support an optional destination hint where we pass "sip:jon@example.org"

3.2. Fingerprint format.

Unify the format of the fingerprints between STIR and WebRTC.

Proposal move to something more like

```
{
  "alg":"HS256",
  "digest":"23:E8:6E:28:3F:D1:CF:17:DE:88:0F:EE:5A:AB:
         AD:03:2D:77:FB:05:17:BA:61:12:1B:37:D4:19:4F:38:DF:EC"
}
```

And perhaps choose a more compact for the digest value - preferably base 64.

3.3. Format of identity header

3.3.1. Token type in Identity header to allow multiple identity headers

Have the type of identity token in a standard form outside the actual token so that things that don't understand the token still know what it is

So the Identity line would change from something like

```
a=identity:eyJhc3NlNzcG9ydHYxIn19;info=URL
```

to

```
a=identity:eyJhc3NlNzcG9ydHYxIn19;info=URL;type=passport
```

3.3.2. WebRTC support multiple tokens

Nice to allow multiple IdP that produce different token types for a given offer in WebRTC.

If the SDP had multiple tokens of different types, do we need WebRTC to be able to support this.

3.3.3. Abstract the identity and protocol in the WebRTC assertion

Currently the WebRTC token has to have the domain and protocol in a well defined location in the assertion. But when this is represented into SDP, it might be better to have it as optional tags on the Identity line instead of in the actual token. So the Identity line would change from something like

```
a=identity:eyJhc3NlNzcG9ydHYxIn19
```

to

```
a=identity:eyJhc3NlNzcG9ydHYxIn19;
        domain=https://ks.fluffy.im:10443;protocol=passport
```

This would allow systems using passport identity to have the same SDP regardless of it the SDP was for WebRTC or SIP

This would also Passport Tokens generated in SIP to work for WebRTC

3.4. Future Proofing

WebRTC allows future things to be added to the contents object passed to the IdP JS. They have to be conveyed to the far side (see section 5.6.4 of [I-D.ietf-rtcweb-security-arch]).

Proposal. Copy them into the claims in the passport token.

3.5. TEL URI

For single phone numbers, we already have a way to encode that in certificates using the SAN with a URI containing a tel URI.

Proposal. Allow existing cert to be valid STIR certificates.

3.6. RFC822 Names

Consider an WebRTC endpoint that initiates a call using WebRTC that is then translated to SIP then translated to XMPP to deliver it to the far end user. Using a name like "sip:fluffy@example.com" is

pretty weird at the WebRTC and XMPP end. It might be better to use 822 style names instead of URIs.

If this was done, it would likely need to use the work that extends the current X509 specifications to allow i18n names. (For example see [I-D.lbaudoin-iemax])

3.7. Encoding

We start with a fingerprint in binary and encoding it like "AA:BB:CC" resulting in an expansion of 3 times. It then expands 1.3 times in the base64 encoding of the payload when then expands 1.3 times in the base64 encoding of the identity header making it over 5 times larger than the original.

3.8. WebRTC Canonical Form

Ensure the definition of what goes in the WebRTC assertion content array is well specified. Ensure it does not need to be bitwise exact but only the same when compared at JS object level.

3.9. Multi party

For a conference call or group MMS, we need to allow multiple destinations.

Proposal: making the duri a list.

4. Acknowledgments

Thank you to Russ Housley for comments.

5. References

5.1. Normative References

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-11 (work in progress), March 2015.

[I-D.ietf-stir-certificates]
Peterson, J., "Secure Telephone Identity Credentials: Certificates", draft-ietf-stir-certificates-03 (work in progress), March 2016.

[I-D.ietf-stir-passport]
Wendt, C. and J. Peterson, "Persona Assertion Token", draft-ietf-stir-passport-01 (work in progress), March 2016.

[I-D.ietf-stir-rfc4474bis]
Peterson, J., Jennings, C., Rescorla, E., and C. Wendt, "Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-stir-rfc4474bis-08 (work in progress), March 2016.

5.2. Informative References

[I-D.lbaudoin-iemax]
Baudoin, L., Chuang, W., and N. Lidzborkski, "Internationalized Electronic Mail Addresses in RFC5280 / X.509 Certificates", draft-lbaudoin-iemax-02 (work in progress), February 2016.

Appendix A. Example Cert

The cert used to generat the examples is:

```
-----BEGIN CERTIFICATE-----
MIIDOzCCAiOgAwIBAgIJANgkZSg8VNNEMA0GCSqGSIb3DQEBCwUAMEExCzAJBgNV
BAYTAkNBMQ8wDQYDVQQIDAZBbGJydGExEDAOBgNVBAcMB0NhbGdhcnkxDzANBgNV
BAsMBkZsdWZmeTAeFw0xNjA1MDIyMjUxMjZaFw0yNjA0MzAyMjUxMjZaMEExCzAJ
BgNVBAYTAkNBMQ8wDQYDVQQIDAZBbGJydGExEDAOBgNVBAcMB0NhbGdhcnkxDzAN
BgNVBAsMBkZsdWZmeTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAKhn
+91RRdt29jwXrUwkAthRA2LQ7UP5bfEy8CpJcgNHEQDryoLQJmQQsP0Hi5qf6CJA
dWW3+vmfKZHhbsDblI8iVCMhekKr/kbplMLSJFdNEkg/cWM3BKBVuqMmmArdpki9
62TmHkO2g+Fk+nubmsB8EDhXvp+TLO1mf1mT6vIUW+U5WVTeghLz8YO+kUjrS0fW
+GWD2lrgEMwvHfuUeBsTD48elXDDPhRAFrB5+TF044Y5+F1lS4iqpD7d1e2rw/YQ
VrHOTJot5T6H33vSvVGGtTZqqDKdTDhCGISb95/IDYSMGT67OoJSlG6/VDyI643V
g0tXkEN2N+TWBv+zDT0CAwEAAaM2MDQwCQYDVR0TBAIwADALBgNVHQ8EBAMCBeAw
GgYDVR0RBBMwEYYPdGVsOjE0MDg0MjE5OTkwMA0GCSqGSIb3DQEBCwUAA4IBAQBT
g/mt3vzs7pTsFCTgNA49vPae+M4IPPqgqfuGPzpCPhzYYnly5EiZYoSaFtYJJiUA
6Hwna0n7SVZdLwHEx7uOk3HHEdjAw8RIf4xSSdjv+IWDDyLl5fLVxyPqnJAAvQ9k
bOV25stTbbKz0EEXV/VKe7A5bILxc2e+JUKjarff2aAEfO8mk+fSVsn5DnBJk3kS
kn8oGaYy5T+5HjdanOfR9ZIZBwTi7XIieSHUzuA7ax5QNugLEgUDPsus1BBxr5Bd
A/HwL0pU7iO4/Hsx1QeNBXy/34yC4n5OCFI5FmfOB+VmJWbDmilH6+uByj339Ii2
2zMX9jtye/So5f5PGlRE
-----END CERTIFICATE-----
```

which contains

```
Certificate:
    Data:
```

```
            Version: 3 (0x2)
            Serial Number:
                d8:24:65:28:3c:54:d3:44
        Signature Algorithm: sha256WithRSAEncryption
            Issuer: C = CA, ST = Albrta, L = Calgary, OU = Fluffy
            Validity
                Not Before: May  2 22:51:26 2016 GMT
                Not After : Apr 30 22:51:26 2026 GMT
            Subject: C = CA, ST = Albrta, L = Calgary, OU = Fluffy
            Subject Public Key Info:
                Public Key Algorithm: rsaEncryption
                    Public-Key: (2048 bit)
                    Modulus:
                        00:a8:67:fb:dd:51:45:db:76:f6:3c:17:ad:4c:24:
                        02:d8:51:03:62:d0:ed:43:f9:6d:f1:32:f0:2a:49:
                        72:03:47:11:00:eb:ca:82:d0:26:64:10:b0:fd:07:
                        8b:9a:9f:e8:22:40:75:65:b7:fa:f9:9f:29:91:e1:
                        6e:c0:db:94:8f:22:54:23:21:7a:42:ab:fe:46:e9:
                        94:c2:d2:24:57:4d:12:48:3f:71:63:37:04:a0:55:
                        ba:a3:26:98:0a:dd:a6:48:bd:eb:64:e6:1e:43:b6:
                        83:e1:64:fa:7b:9b:9a:c0:7c:10:38:57:be:9f:93:
                        2c:ed:66:7f:59:93:ea:f2:14:5b:e5:39:59:54:de:
                        82:12:f3:f1:83:be:91:48:eb:4b:47:d6:f8:65:83:
                        da:5a:e0:10:cc:2f:1d:fb:94:78:1b:13:0f:8f:1e:
                        95:70:c3:3e:14:40:16:b0:79:f9:31:74:e3:86:39:
                        f8:5d:65:4b:88:aa:a4:3e:dd:d5:ed:ab:c3:f6:10:
                        56:b1:ce:4c:9a:2d:e5:3e:87:df:7b:d2:bd:51:86:
                        b5:36:6a:a8:32:9d:4c:38:42:18:84:9b:f7:9f:c8:
                        0d:84:8c:19:3e:bb:3a:82:52:94:6e:bf:54:3c:88:
                        eb:8d:d5:83:4b:57:90:43:76:37:e4:d6:06:ff:b3:
                        0d:3d
                    Exponent: 65537 (0x10001)
            X509v3 extensions:
                X509v3 Basic Constraints:
                    CA:FALSE
                X509v3 Key Usage:
                    Digital Signature, Non Repudiation, Key Encipherment
                X509v3 Subject Alternative Name:
                    URI:tel:14084219990
        Signature Algorithm: sha256WithRSAEncryption
             53:83:f9:ad:de:fc:ec:ee:94:ec:14:24:e0:34:0e:3d:bc:f6:
             9e:f8:ce:08:3c:fa:a0:a9:fb:86:3f:3a:42:3e:1c:d8:62:79:
             72:e4:48:99:62:84:9a:16:d6:09:26:25:00:e8:7c:27:6b:49:
             fb:49:56:5d:2f:01:c4:c7:bb:8e:93:71:c7:11:d8:c0:c3:c4:
             48:7f:8c:52:49:d8:ef:f8:85:83:0f:22:e5:e5:f2:d5:c7:23:
             ea:9c:90:00:bd:0f:64:6c:e5:76:e6:cb:53:6d:b2:b3:d0:41:
             17:57:f5:4a:7b:b0:39:6c:82:f1:73:67:be:25:42:a3:6a:b7:
             df:d9:a0:04:7c:ef:26:93:e7:d2:56:c9:f9:0e:70:49:93:79:
```

```
            12:92:7f:28:19:a6:32:e5:3f:b9:1e:37:5a:9c:e7:d1:f5:92:
            19:07:04:e2:ed:72:22:79:21:d4:ce:e0:3b:6b:1e:50:36:e8:
            0b:12:05:03:3e:cb:ac:d4:10:71:af:90:5d:03:f1:f0:2f:4a:
            54:ee:23:b8:fc:7b:31:d5:07:8d:05:7c:bf:df:8c:82:e2:7e:
            4e:08:52:39:16:67:ce:07:e5:66:25:66:c3:9a:29:47:eb:eb:
            81:ca:3d:f7:f4:88:b6:db:33:17:f6:3b:72:7b:f4:a8:e5:fe:
            4f:1a:54:44
```

Appendix B. Example Private Key

The private key for the cert is:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAqGf73VFF23b2PBetTCQC2FEDYtDtQ/lt8TLwKklyA0cRAOvK
gtAmZBCw/QeLmp/oIkB1Zbf6+Z8pkeFuwNuUjyJUIyF6Qqv+RumUwtIkV00SSD9x
YzcEoFW6oyaYCt2mSL3rZOYeQ7aD4WT6e5uawHwQOFe+n5Ms7WZ/WZPq8hRb5TlZ
VN6CEvPxg76RSOtLR9b4ZYPaWuAQzC8d+5R4GxMPjx6VcMM+FEAWsHn5MXTjhjn4
XWVLiKqkPt3V7avD9hBWsc5Mmi3lPoffe9K9UYa1NmqoMp1MOEIYhJv3n8gNhIwZ
Prs6glKUbr9UPIjrjdWDS1eQQ3Y35NYG/7MNPQIDAQABAoIBAF6ELdmi+aAY/k3v
w/WN6ILbxRi6xc92uHu86QnyuqiYRDTOIZSVmlZi/9KjX3ji8nf20WzLe3KKH9ye
N3jKRHCpBavJ6EJvIYFPK4zEQF03BmHCKbNTd6c9NkjHKmI+0ErXPLweYzIBx7bC
48poJMyPVNMqe/Q3t+lts1/lIuHGHhdnSn7ao6hP+xE7V+PMXic1f/hL7WwpCUvt
N0IQp4sgqbrZwfVaGtkDSrzdYELYQb9QHEiMGoYg6oC7uGMoX1zi5B52BRP8MArA
9htEx4r9bAFCBX/lvQkq3QP5Pg6okhAB2eqEk2zUzUJlfHC1kr6aw8744potJmJW
ZiOdzkECgYEA2S8aw+CI6h9T07BBbX+xFqQhUot0ngeErW49BgvrOLwLEacI9ofb
b4BkNGzcFMejAk2fqRfhwwu53Y8+5u+xMrw3pC/lvrDjFNaXD3nILr1PdlAE/hee
hMkb60HQFYSoG6ZcdbqSeYkp1fn5TsN7tAhQT21NVxEWb4J9az02LrkCgYEAxoEh
gfczbkFM/dDiCXg2R9LxKTNwiZD2iugVLV0kML4zJryk5/QVK5wumOtZpjfJzVi3
hC0jL7gzhHUymfo6owG72S5qiodbdkhHqpqODVlCLgY+3VTBBiomLO9YWSYzW6QO
sOQCA+dwBuL/6IEY4rFX6x/AL8sh4JSXYjcGcKUCgYEA0FKVmtOyoNgh4UkMyUqV
hAE1kWcBGmBdzLmUQUuHeikteOY++7K/Mon2FC9jP29rFdd9UYX94MhLpZE0pfG+
h8rwmEX1Wt9zQlbAGXEYKnUeVn9U+qGPRRFe/V9oiGtxkOwXfjnTLE78WSppEDsE
WmErH7TZXa2fVqDVStsxMMkCgYBMgPQTDNzLf2tW3yxejfANmmTLhkG3IyGBw5R1
2VHbX1KDeWzs4ItQNW9YDEyO3S1vcOO5k1PeTlW8lRaddW0n6cEmINd68FP1sEG+
pLZeuqng5xNPZhzGbXQtGUmpgimFBiOLVTTZoFbysIYEa8zVgZfqzF/bi6RQ07PM
bHyU6QKBgQDYCytiEWCz7vKoSRjE5vng3eNRVtYbbe8AiJv6qjYz8rOosJEMDoMV
vPOREdf2EkXdOODvgqCjh0HDCP112qyoMoQB/PC9ihum4ZTZn1KUI5Rj0GHc1GwW
d8jDOR91y0tjvnLQv37RLe4j9327ism0cf8NSlMe4tsWJocF6JYNoA==
-----END RSA PRIVATE KEY-----
```

Author's Address

Cullen Jennings
Cisco Systems

Email: fluffy@iii.ca