Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2019

N. Sullivan
Cloudflare
H. Krawczyk
IBM Research
O. Friel
R. Barnes
Cisco

March 11, 2019

# Usage of OPAQUE with TLS 1.3
# draft-sullivan-tls-opaque-00

## Abstract

This document describes two mechanisms for enabling the use of the OPAQUE password-authenticated key exchange in TLS 1.3.

## 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 September 12, 2019.

## Copyright Notice

Table of Contents



## 1. Introduction

Note that this draft has not received significant security review and should not be the basis for production systems.

OPAQUE [opaque-paper] is a mutual authentication method that enables the establishment of an authenticated cryptographic key between a client and server based on a user's memorized password, without ever exposing the password to servers or other entities other than the client machine and without relying on PKI. OPAQUE leverages a primitive called a Strong Asymmetrical Password Authenticated Key Exchange (Strong aPAKE) to provide desirable properties including resistance to pre-computation attacks in the event of a server compromise.

In some cases, it is desirable to combine password-based authentication with traditional PKI-based authentication as a defense-in-depth measure. For example, in the case of IoT devices, it may be useful to validate that both parties were issued a certificate from a certain manufacturer. Another desirable property





for password-based authentication systems is the ability to hide the client's identity from the network. This document describes the use of OPAQUE in TLS 1.3 [TLS13] both as part of the TLS handshake and post-handshake facilitated by Exported Authenticators [I-D.ietf-tls-exported-authenticator], how the different approaches satisfy the above properties and the trade-offs associated with each design.

The in-handshake instantiations of OPAQUE can be used to authenticate a TLS handshake with a password alone, or in conjunction with certificate-based (mutual) authentication but does not provide identity hiding for the client. The Exported Authenticators instantiation of OPAQUE provides client identity hiding by default and allows the application to do password authentication at any time during the connection, but requires PKI authentication for the initial handshake and application-layer semantics to be defined for transporting authentication messages.

## 2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 3. OPAQUE

In OPAQUE [opaque-paper], it is shown that a Strong Asymmetric Password-Authenticated Key Exchange (Strong aPAKE) can be constructed given an oblivious pseudo-random function (OPRF) and authenticated key exchange protocol that is secure against reverse impersonation (a.k.a. KCI). Unlike previous PAKE methods such as SRP [RFC2945] and SPAKE-2 [I-D.irtf-cfrg-spake2], which require a public salt value, a Strong aPAKE leverages the OPRF private key as salt, making it resistant to pre-computation attacks on the password database

stored on the server.

TLS 1.3 provides a KCI-secure key agreement algorithm suitable for use with OPAQUE. This document describes three instantiations of OPAQUE in TLS 1.3: one based on digital signatures, one on Diffie-Hellman key agreement, and one based on HMQV key exchange. Of the three instantiations, the only one that has known IPR considerations is HMQV.

OPAQUE consists of two distinct phases: password registration and authentication. We will describe the mechanisms for password registration in this document but it is assumed to have been done





outside of TLS. During password registration, the client and server establish a shared set of parameters for future authentication and two private-public key pairs are generated, one for the client and one for the server. The server keeps its private key and stores an encapsulated copy of the client's key pair along with its own public key in an "envelope" that is encrypted with the result of the OPRF operation. Note that it is possible for the server to use the same key for multiple clients. It may be necessary to permit multiple simultaneous server keys in the even of a key rollover. The client does not store any state nor any PKI information.

We call the first instantiation OPAQUE-Sign. In OPAQUE-Sign, the key pairs generated at password registration time are digital signature keys. These signature keys are used in place of certificate keys for both server and client authentication in a TLS handshake. Client authentication is technically optional, but in practice is almost universally required. OPAQUE-Sign cannot be used alongside certificate-based handshake authentication. This instantiation can also be leveraged to do part of a post-handshake authentication using Exported Authenticators [I-D.ietf-tls-exported-authenticator] given an established TLS connection protected with certificate-based authentication.

The second and third instantiations are called OPAQUE-3DH and OPAQUE-HMQV. In these instantiations, the key pairs are Diffie-Hellman keys and are used to establish a shared secret that is fed into the key schedule for the handshake. The handshake continues to use Certificate-based authentication. The two methods for establishing the shared key are Diffie-Hellman and HMQV. These instantiations are

best suited to use cases in which both password and certificate-based authentication are needed during the initial handshake, which is useful in some scenarios. There is no unilateral authentication in this context, mutual authentication is demonstrated explicitly through the finished messages.

## 4. Password Registration

Password registration is run between a user U and a server S. It is assumed that the user can authenticate the server during this registration phase (this is the only part in OPAQUE that requires some form of authenticated channel, either physical, out-of-band, PKI-based, etc.)

A set of parameters is chosen. This includes an AuthEnc function for key encapsulation, a group setting for the OPRF (chosen as a cipher defined in Oblivious Pseudorandom Functions (OPRFs) using Prime-Order Groups [I-D.sullivan-cfrg-voprf]), an instantiation (either OPAQUE-Sign, OPAQUE-3DH or OPAQUE-HMQV), and a key type (either a TLS





Signature Scheme [TLS13] for OPAQUE-Sign or a TLS Supported Group [TLS13] for OPAQUE-3DH and OPAQUE-HMQV).

- U chooses password PwdU and a pair of private-public keys PrivU and PubU of the chosen key type.
- S chooses OPRF key kU (random and independent for each user U) and sets vU = g^kU; it also chooses its own pair of private-public keys PrivS and PubS (the server can use the same pair of keys with multiple users), and sends PubS to U.
- U and S run OPRF(kU;PwdU) as defined in with only U learning the result, denoted RwdU (mnemonics for "Randomized PwdU").
- U generates an "envelope" EnvU defined as

EnvU = AuthEnc(RwdU; PrivU, PubU, PubS)

where AuthEnc is an authenticated encryption function with the "key committing" property and is specified below in section. In EnvU, all values require authentication and PrivU also requires encryption. PubU can be omitted from EnvU if it can be reconstructed from PrivU

but while it will save bits on the wire it will come at some computational cost during client authentication.

- U sends EnvU and PubU to S and erases PwdU, RwdU and all keys. S stores (EnvU, PubS, PrivS, PubU, kU, vU) in a user-specific record. If PrivS and PubS are used for multiple users, S can store these values separately and omit them from the user's record.

Note (salt). We note that in OPAQUE the OPRF key acts as the secret salt value that ensures the infeasibility of pre-computation attacks. No extra salt value is needed.

## 4.1. Implementing EnvU

The encryption for EnvU is required to be a key-committing authenticated encryption algorithm. This, unfortunately, eliminates both AES-GCM and AES-GCM-SIV as wrapping functions. It is possible to create a key-committing authenticated encryption using AES-CBC [RFC3602] or AES-CTR [RFC5930] with HMAC [RFC4868] as long as the keys for encryption and authentication are derived separately with a key domain separation mechanism such as HKDF [RFC5869].





# 5. TLS extensions

We define several TLS extensions to signal support for OPAQUE and transport the parameters. The extensions used here have a similar structure to those described in Usage of PAKE with TLS 1.3 [I-D.barnes-tls-pake]. The permitted messages that these extensions are allowed and the expected protocol flows are described below.

This document defines the following extension code points.

```
enum {
  ...
  opaque_client_auth(TBD),
  opaque_server_auth(TBD),
  (65535)
```

```
} ExtensionType;
```

The opaque_client_auth extension contains a PAKEClientAuthExtension struct and can only be included in the CertificateRequest and Certificate messages. The opaque_client_auth extension contains a PAKEServerAuthExtension struct and can only be included in the ClientHello, EncryptedExtensions, CertificateRequest and Certificate messages, depending on the type.

The structures contained in this extension are defined as:

```
struct {
  opaque identity<0..2^16-1>;
  opaque OPRF_1<1..2^16-1>;
} PAKEShareClient;
```





```
struct {
 opaque identity<0..2^16-1>;
 opaque OPRF_2<1..2^16-1>;
 opaque vU<1..2^16-1>;
 opaque EnvU<1..2^16-1>;
} PAKEShareServer;

struct {
```

```
      select (Handshake.msg_type) {
        ClientHello:
          PAKEShareClient client_shares<0..2^16-1>;
          OPAQUEType types<0..2^16-1>;
        EncryptedExtensions, Certificate:
          PAKEShareServer server_share;
          OPAQUEType type;
      }
    } PAKEServerAuthExtension;

    struct {
      opaque identity<0..2^16-1>;
    } PAKEClientAuthExtension;
```

This document also defines the following set of types;

```
    enum {
      OPAQUE-Sign(1),
      OPAQUE-3DH(2),
      OPAQUE-3DH-Cert(3),
      OPAQUE-HMQV(4),
      OPAQUE-HMQV-Cert(5),
    } OPAQUEType;
```

The "identity" field is the unique user id used to index the user's record on the server. The types field indicates the set of supported auth types by the client. The OPRF_1 message is as defined in Oblivious Pseudorandom Functions (OPRFs) using Prime-Order Groups [I-D.sullivan-cfrg-voprf]. The content of OPRF_1 is typically the result of the password hashed into a group element and blinded by an element known to the client. OPRF_2 is the OPRF_1 value operated on by the OPRF private key kU. vU is the public component of kU and EnvU is the envelope containing PrivU, PubS, and PubU. (Note that for groups, it may be more space efficient to only include PrivU and have the client derive PubU from PrivU). See Section 9 for details.

This document also describes a new CertificateEntry structure that corresponds to an authentication via a signature derived using OPAQUE. This structure serves as a placeholder for the PAKEServerAuthExtension extension.





```
    struct {
```

```
      select (certificate_type) {
        case OPAQUESign:
          /* Defined in this document */
          opaque null<0>

        case RawPublicKey:
          /* From RFC 7250 ASN.1_subjectPublicKeyInfo */
          opaque ASN1_subjectPublicKeyInfo<1..2^24-1>;

        case X509:
          opaque cert_data<1..2^24-1>;
      };
      Extension extensions<0..2^16-1>;
    } CertificateEntry;
```

We request that IANA add an additional type to the "TLS Certificate Types" registry for this OPAQUESign.

Support for the OPAQUESign Certificate type for server authentication can be negotiated using the server_certificate_type [RFC7250] and the Certificate type for client authentication can be negotiated using the client_certificate_type extension [RFC7250].

Note that there needs to be a change to the client_certificate_type row in the IANA TLS ExtensionType Values table to allow client_certificate_type extension to be used as an extension to the CertificateRequest message.

# 6. Use of extensions in TLS handshake flows

## 6.1. OPAQUE-3DH, OPAQUE-HMQV

In these two modes of operation, the OPAQUE private keys are used for key agreement algorithm and the result is fed into the TLS key schedule. Password validation is confirmed by the validation of the finished message. These modes can be used in conjunction with optional Certificate-based authentication.

It should be noted that since the identity of the client it is not encrypted as it is sent as an extension to the ClientHello. This may present a privacy problem unless a mechanism like ESNI [I-D.ietf-tls-esni] is created to protect it.

Upon receiving a PAKEServerAuth extension, the server checks to see if it has a matching record for this identity. If the record does not exist, the handshake is aborted with a TBD error message. If the record does exist, but the key type of the record does not match any

of the supported_groups sent in the key_share extension of the ClientHello, an HRR is sent containing the set of valid key types that it found records for.

Given a matching key_share and an identity with a matching supported_group, the server returns its PAKEServerAuth as an extension to its EncryptedExtensions. Both parties then derive a shared OPAQUE key using

HMQV

```
   C computes K = (g^y * PubS^e)^{x + d*PrivU)
   S computes K = (g^x * PubU^d)^{y + e*PrivS
```

where d = H(g^x, IdS) and e = H(g^y, IdU), and IdU, IdS represent the identities of user (sent as identity in PAKEShareClient) and server (EncryptedExtension or Certificate message). TODO: be more explicit about content of IdS.

3DH

```
C computes K = H(g^y ^ PrivU || PubU ^ x || PubS ^ PrivU || IdU || IdS )
S computes K = H(g^x ^ PrivS || PubS ^ y || PubU ^ PrivS || IdU || IdS )
```

IdU, IdS represent the identities of user (sent as identity in PAKEShareClient) and server (Certificate message).

H is the HKDF function agreed upon in the TLS handshake.

The result, K, is then added as an input to the Master Secret in place of the 0 value defined in TLS 1.3:

```
  0 -> HKDF-Extract = Master Secret
```

becomes

```
  K -> HKDF-Extract = Master Secret
```

In this construction, the finished messages cannot be validated unless the OPAQUE computation was done correctly on both sides, authenticating both client and server.

For the certificate version of OPAQUE (OPAQUE-3DH-Cert, OPAQUE-HMQV-Cert), the server's first flight contains the standard set of messages: ServerHello, EncryptedExtension, (optional)CertificateRequest, Certificate, CertificateVerify,

Finished. In the non-certificate cases (OPAQUE-3DH-Cert, OPAQUE-





HMQV-Cert), the Certificate and CertificateVerify messages are omitted, similar to the PSK mode in TLS 1.3.

## 6.2. OPAQUE-Sign

In this modes of operation, the OPAQUE private keys are used for digital signatures and are used to define a new Certificate type and CertificateVerify algorithm. Like the 3DH and HKDF instantiations above, the identity of the client is sent in the clear in the client's first flight unless a mechanism like ESNI [I-D.ietf-tls-esni] is created to protect it.

Upon receiving a PAKEServerAuth extension, the server checks to see if it has a matching record for this identity. If the record does not exist, the handshake is aborted with a TBD error message. If the record does exist, but the key type of the record does not match any of the supported_signatures sent in the the ClientHello, the handshake must be aborted with a TBD error.

We define a new Certificate message type for an OPAQUE-Sign authenticated handshake.

```
enum {
  X509(0),
  RawPublicKey(2),
  OPAQUE-Sign(3),
  (255)
} CertificateType;
```

Certificates of this type have CertificateEntry structs of the form:

```
struct {
  Extension extensions<0..2^16-1>;
} CertificateEntry;
```

Given a matching signature_scheme and an identity with a matching key type, the server returns a certificate message with type OPAQUE-Sign with PAKEServerAuth as an extension. The private key used in the CertificateVerify message is set to PrivS, and the client verifies it

using PubS.

It is RECOMMENDED that the server includes a CertificateRequest message with a PAKEClientAuth and the identity originally sent in the PAKEServerAuth extension from the client hello. On receiving a CertificateRequest message with a PAKEClientAuth extension, the client returns a CertificateVerify message signed by PrivC which is validated by the server using PubC.





## 7. Integration into Exported Authenticators

Neither of the above mechanisms provides privacy for the user during the authentication phase, as the user id is sent in the clear. It is possible to create an encryption mechanism like ESNI [I-D.ietf-tls-esni] to protect these values, but this is not in scope for this document. Additionally, OPAQUE-Sign has the drawback that it cannot be used in conjunction with certificate-based authentication.

It is possible to address both the privacy concerns and the requirement for certificate-based authentication by using OPAQUE-Sign in Exported Authenticator [I-D.ietf-tls-exported-authenticator] flow, since exported authenticators are sent over a secure channel that is typically established with certificate-based authentication. Using Exported Authenticators for OPAQUE has the additional benefit that it can be triggered at any time after a TLS session has been established, which better fits modern web-based authentication mechanism.

The client hello contains PAKEServerAuth, PAKEClientAuth with empty identity values to indicate support for these mechanisms.

1. Client creates Authenticator Request with CR extension PAKEServerAuth (identity, OPRF_1)

2. Server creates Exported Authenticator with OPAQUE-Sign (PAKEServerAuth) and CertificateVerify (signed with PrivS)

If the server would like to then establish mutual authentication, it can do the following: 1. Server creates Authenticator Request with CH extension PAKEClientAuth (identity) 2. Client creates Exported

Authenticator with OPAQUE-Sign Certificate and CertificateVerify (signed with PrivU)

Support for Exported Authenticators is negotiated at the application layer. For example, OPAQUE-Sign in EAs could be defined as an extension to Secondary Certificates in HTTP/2 [I-D.ietf-httpbis-http2-secondary-certs].

## 8. Summary of properties





| Variant \ Property | Identity Hiding | Certificate Authentication | Server-only Auth | Post-handshake auth | Minimum round trips |
|---|---|---|---|---|---|
| OPAQUE-Sign-EA | yes | yes | yes | yes | 2-RTT |
| OPAQUE-Sign | no | no | yes | no | 1-RTT |
| OPAQUE-3DH | no | no | no | no | 1-RTT |
| OPAQUE-3DH-Cert | no | yes | no | no | 1-RTT |
| OPAQUE-HMQV | no | no | no | no | 1-RTT |
| OPAQUE-HMQV-Cert | no | yes | no | no | 1-RTT |

## 9. Example OPRF

This is an example OPRF instantiation based on the ECOPRF-P256-HKDF-

SHA256-SSWU ciphersuite in [I-D.sullivan-cfrg-voprf]. We use additive group notation in this description because we specifically target the elliptic curve case. All operations can be replaced with their multiplicative group counterparts.

The example ECOPRF-P256-HKDF-SHA256-SSWU instantiation uses the following parameters:

- Curve: SECP256K1 curve
- H_1: H2C-P256-SHA256-SSWU- [I-D.sullivan-cfrg-voprf]
- label: voprf_h2c
- H_2: SHA256

See [I-D.sullivan-cfrg-voprf] for more details about how each of the above components are used. In the following we will use the functions OPRF_Blind, OPRF_Sign, OPRF_Unblind, OPRF_Finalize that are defined in the same document.





### 9.1. OPRF_1

Let p be the prime order of the base field of the curve that is used (e.g. 2^256 - 2^224 + 2^192 + 2^96 - 1 for P-256). Let I2OSP, OS2IP be functions as defined in [RFC8017]. Then OPRF_1 is computed using the OPRF_Blind function on the password P follows:

1. r <-$ GF(p)
2. M := rH_1(P)
3. Output (r, M)

H_1 = hash-to-curve(P) = 1. t1 = H("h2c" || label || I2OSP(len(x), 4) || P) 2. t2 = OS2IP(t1) 3. y = t^2 mod p 4. H_1(P) = map2curve_simple_swu(y) 5. M = rH_1(P)

The client keeps the blind r, and sends the OPRF_1 value M as an

EllipticCurve point [TLS13].

## 9.2. OPRF_2

The server now computes OPRF_2 by applying OPRF_Sign on the received message M: 1. Z := kM 2. Output Z Note that the server should multiply M by the cofactor of the given curve before it outputs Z. In the case of P-256, this cofactor is equal to 1 and so it is not necessary.

The output Z of OPRF_2 is sent as an EllipticCurve point "[]" back to the client.

When the client receives the output of OPRF_2, it derives the envelope decryption key using OPRF_Unblind followed by OPRF_Finalize.

1. N := (1/r)Z (OPRF_Unblind)

2. y := H_2(P, N) (OPRF_Finalize). Here, we require that N is serialized before it is input to H_2. The client can now stores (P, y) for future usage.

# 10. Privacy considerations

TBD





# 11. Security Considerations

TODO: protecting against user enumeration

# 12. IANA Considerations

- Existing IANA references have not been updated yet to point to this document.

  IANA is asked to register a new value in the "TLS Certificate Types" registry of Transport Layer Security (TLS) Extensions (TLS-

Certificate-Types-Registry), as follows:

- Value: 4 Description: OPAQUE Authentication Reference: This RFC

Correction request: The client_certificate_type row in the IANA TLS ExtensionType Values table to allow client_certificate_type extension to be used as an extension to the CertificateRequest message.

# 13. References

## 13.1. Normative References


[I-D.ietf-httpbis-http2-secondary-certs]
Bishop, M., Sullivan, N., and M. Thomson, "Secondary Certificate Authentication in HTTP/2", draft-ietf-httpbis-http2-secondary-certs-03 (work in progress), October 2018.

[I-D.ietf-tls-exported-authenticator]
Sullivan, N., "Exported Authenticators in TLS", draft-ietf-tls-exported-authenticator-08 (work in progress), October 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3602] Frankel, S., Glenn, R., and S. Kelly, "The AES-CBC Cipher Algorithm and Its Use with IPsec", RFC 3602, DOI 10.17487/RFC3602, September 2003, <https://www.rfc-editor.org/info/rfc3602>.

[RFC4868] Kelly, S. and S. Frankel, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec", RFC 4868, DOI 10.17487/RFC4868, May 2007, <https://www.rfc-editor.org/info/rfc4868>.







[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250,

June 2014, <https://www.rfc-editor.org/info/rfc7250>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

## 13.2. Informative References

[I-D.barnes-tls-pake]
Barnes, R. and O. Friel, "Usage of PAKE with TLS 1.3", draft-barnes-tls-pake-04 (work in progress), July 2018.

[I-D.ietf-tls-esni]
Rescorla, E., Oku, K., Sullivan, N., and C. Wood, "Encrypted Server Name Indication for TLS 1.3", draft-ietf-tls-esni-03 (work in progress), March 2019.

[I-D.irtf-cfrg-spake2]
Ladd, W. and B. Kaduk, "SPAKE2, a PAKE", draft-irtf-cfrg-spake2-08 (work in progress), March 2019.

[I-D.sullivan-cfrg-voprf]
Davidson, A., Sullivan, N., and C. Wood, "Oblivious Pseudorandom Functions (OPRFs) using Prime-Order Groups", draft-sullivan-cfrg-voprf-03 (work in progress), March 2019.

[opaque-paper]
Xu, J., "OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks", 2018.

[RFC2945] Wu, T., "The SRP Authentication and Key Exchange System", RFC 2945, DOI 10.17487/RFC2945, September 2000, <https://www.rfc-editor.org/info/rfc2945>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC5930] Shen, S., Mao, Y., and NSS. Murthy, "Using Advanced Encryption Standard Counter Mode (AES-CTR) with the Internet Key Exchange version 02 (IKEv2) Protocol", RFC 5930, DOI 10.17487/RFC5930, July 2010, <https://www.rfc-editor.org/info/rfc5930>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/info/rfc8017>.

## Appendix A. Acknowledgments

## Authors' Addresses

Nick Sullivan
Cloudflare

Email: nick@cloudflare.com

Hugo Krawczyk
IBM Research

Email: hugo@ee.technion.ac.il

Owen Friel
Cisco

Email: ofriel@cisco.com

Richard Barnes
Cisco

Email: rlb@ipv.sx