Hybrid Public Key Encryption (HPKE) with deterministically generated key pairs using Tink

Asked 11 months ago Modified 11 months ago Viewed 822 times

0

I would like to use Google Tink for public key encryption / private key decryption, mostly because it's high-level enough and is available in a variety of programming languages (e.g. from Android and IOS).

I chose keys of type `DHKEM_X25519_HKDF_SHA256_HKDF_SHA256_AES_256_GCM` which use X25519 Elliptic Curve Cryptography, because they are:

- recommended by https://developers.google.com/tink/hybrid
- not using a curve recommended by NIST

I need those key to be generated deterministically, for disaster recovery. There are other layers of security, and my main concern is data loss.

Private key generation is under control (!). I think public key are generated by Tink on-the-fly, as it's a simple arithmetical operation given the private key. However, I can't create a `HpkePrivateKey` in java without providing both public and private keys (and using an API in alpha).

I also tried to hack my way into importing hand-made JSON with `JsonKeysetReader`, but I get not-so-verbose errors about my key data not matching the expected ProtoBuf format. I'd go for it (at least temporarily) if I could make it work, but honestly I'd feel better implementing something that doesn't feel so much like a hack.

I'm struggling to find a solution to implement this. Any idea on how to import/generate a `KeysetHandle` containing a valid HPKE key pair, given a raw 32-byte private key?

java **public-key-encryption** **private-key** **deterministic** **tink**

Share

Improve this question

Follow

asked Jan 5 at 15:28

Pat Lee

1,658 ● 1 ● 11 ● 14

2 Sorry to disappoint you, but the Bernstein curves (25519 and 448) ARE NIST-approved, as are the related EdDSA signature algorithms, by FIPS186-5 and SP800-186 last Feb. However, SP800-56A has not been updated so at least the XDH key agreements are unapproved, and 140-3 IG C.K resolution 5 explicitly confirms this. :-} :-{
– dave_thompson_085 Jan 6 at 8:54

@dave_thompson_085 good to know. I'll cope with it though, my concern was more about the curves being designed with obscure parameters and potential backdoors. I can trust curves designed by Bernstein.

crypto.stackexchange.com/questions/10263/…
en.wikipedia.org/wiki/Bernstein_v._United_States – Pat Lee
Jan 8 at 8:10

3 Answers

Sorted by: Highest score (default)

3

If you can generate the private key deterministically, then whatever you use to do that **is the key**. You need to protect those parameters just like you would protect the private key. You don't need that deterministic extra layer of complexity.

So what you need to do is a way to **recover a random private key** in case of a disaster. I would just store both the private key and public key pair save all that JSON trouble with Tink. It's public after all. But how do you do that?

Short answer, you do a key ceremony. Watch some ceremonies here, about 4h each.

Fortunately, disasters are rare. The solution will be a mix of

- Your RTO (return to operation) requirements
- Existing secret disaster process you can piggy back on

For example, you could have **the system generate its own private key and output it**, encryted with an escrow's public key. You contact that escrow every year

in your DR test, so that you are ready when a real disaster strikes (if ever).

In a company large enough, there might already be fireproof safes in multiple locations you can use to store a USB key, a CD and a sealed envellope with a printout of the encrypted private key and plain text public key. If you don't have such infrastructure, you can contract a notary to be the escrow.

Another option is to generate a key with a Shamir secret sharing scheme, made popular by Hashicorp Vault. The scheme could be used to generate a key that would decrypt your operation private key.

*You could put the USB and CD under seal too, in case spy cameras get soo good they can read the back of a CD with enough precision to reconstruct it.

Share Improve this answer Follow

edited Jan 7 at 1:16

answered Jan 5 at 20:42

ixe013

10.1k ● 4 ● 50 ● 81

First of all, thank you for the detailed and informative answer! Very interesting, but I think it's way overkill for my use case. Maybe "disaster recovery" was a bad choice of words, sorry. The point is to provide encrypted file sharing for customers, to which even my company wouldn't have access, hence the need to re-generate the private key from user input in case they drop the smartphone (and its key store with the only

copy of the private key) into the sea. – Pat Lee Jan 8 at 8:49

1

In cryptography, if you generate or derive a key from "something", that something is the key. That's because the algorithm is deterministic. It does not have to be, as you could generate a key by sampling noise around you. But in your case, it seems that you want to generate a key from user input.

You'll notice that I haven't mentioned symmetric or asymmetric cryptography yet.

Let's say that user input is a password (better: a passphrase). You said that you want to provide encrypted file sharing, so asymmetric cryptography comes naturally. But generating a key pair involves a good source of entropy, or randomness.

So to enable encrypted file sharing, where a server will not be able to decrypt the files shared between users, you could do the following:

In the browser:

1. Get the passphrase from the user - that's deterministic. Throw in some user specific metadata in the mix so that two users with the same passphrase don't have the same key.
2. Generate a symmetric key from the input+metadata, using Argon2 for example.

3. It is more convenient for the user to save the symmetric key locally, but it increases its exposure (reduces its security). I'll let you figure out the right balance here.
4. Generate a key pair randomly
5. Encrypt the private key with the symmetric key generated at step 2
6. Save the encrypted private key locally
7. Send the encrypted private key and the plain text public key to the server.
8. When someone sends you an encrypted file, decrypt the private key and use it to decrypt the file.

(The actual file encryption will likely be hybrid, where a random symmetric key is encrypted with the recipient's public key, and used to bulk encrypt the file much faster)

Since the symmetric key generated at step 2 never reached the server, the server cannot see the files going through it.

When disaster strikes on the client (or when they just want to read files from another computer):

1. Do steps 1, 2 and 3 in the process as above. You will recover the initial symmetric key.
2. Instead of generating a key pair, request the encrypted private key from the server

3. Decrypt the private key and use it to read files shared with you.

+I suggest that you integrate a key distribution test in the onboarding process.

1. After generating the key pair, encrypt some flag with it. The time in seconds modulo 600 (10 minute increment) for example.
2. Do not save the key just yet. Just send the encrypted private key and public key (step 7 above).
3. Ask the encrypted key from the server and try to decrypt it
4. Decrypt the flag with the key.

The user won't notice the extra round trip, and you'll have confidence that it was saved properly on the server.

Share Improve this answer

Follow

answered Jan 10 at 22:38

ixe013

10.1k ● 4 ● 50 ● 81

That's close enough to what I had in mind. With one main difference. Basically you're suggesting incrementing the entropy of the actual key pair at the cost of having to have the server acting as an encrypted backup key ring, moving the determinism to the generation of a local KEK instead. I'm seriously considering it, but I fail to see how it is worth the added complexity, knowing that all of this happens though channels that are already allegedly secure. Maybe just I need a cryptographer to hit me in the face and shout me that

deterministic key pair generation is a bad idea 😅. Thanks! – Pat Lee Jan 12 at 9:38

Regarding my previous comment, it just came to my mind that one downside of a deterministically generated HPKE key pair is that it tightly links the password and the public key... and eventually, someone's gonna want to change their password. – Pat Lee Jan 12 at 15:21

1 They are not linked. When the password (aka the key generation material) changes, you re-encrypt the **randomly generated private key** and send the new encrypted copy to the server. Bonus points if you implemented and reuse the key distribution test when the change happens. – ixe013 Jan 12 at 17:11

1 Sure, I meant in my original idea of generating the private key of the HPKE key pair deterministically. I'll accept your answer as it circumvents the issues raised by my original question and helps me avoid future issues by decoupling password and public key. Thank you! – Pat Lee Jan 15 at 8:47

0

Here's a solution that doesn't imply messing with Protobuf or JSON...

```
byte[] privateKeyAsBytes = new byte[32];
new SecureRandom().nextBytes(privateKeyAsBytes); // re
for deterministic private key generation

// Required for X25519 private keys, from
com.google.crypto.tink.subtle.X25519::generatePrivateK
privateKeyAsBytes[0] |= 7;
privateKeyAsBytes[31] &= 63;
privateKeyAsBytes[31] |= 128;

byte[] publicKeyAsBytes =
com.google.crypto.tink.subtle.X25519.publicFromPrivate
subtle API, provided but not whose usage is not recomm
```

```
com.google.crypto.tink.hybrid.HpkePublicKey hpkePublic
com.google.crypto.tink.hybrid.HpkePublicKey.create(
        (HpkeParameters)
KeyTemplates.get("DHKEM_X25519_HKDF_SHA256_HKDF_SHA256
        Bytes.copyFrom(publicKeyAsBytes),
        1 // or a random id...
);
com.google.crypto.tink.hybrid.HpkePrivateKey hpkePriva
com.google.crypto.tink.hybrid.HpkePrivateKey.create(
        hpkePublicKey,
        SecretBytes.copyFrom(privateKeyAsBytes, Insecu
);

KeysetHandle keysetHandle =
        KeysetHandle
                .newBuilder()
                .addEntry(
                        KeysetHandle.importKey(hpkePri
                )
                .build();
```

Note: Tink does not compute public keys on-the-fly, they are computed once and then stored alongside the private keys.

Share Improve this answer

Follow

answered Jan 8 at 21:13

Pat Lee

1,658 ● 1 ● 11 ● 14