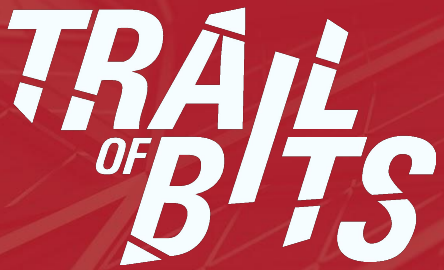


Best Practices for Cryptography in Python

PyCon AU 2019 // August 2, 2019 // Paul Kehrer

Secure cryptography is possible in Python

(restrictions apply)

Who am I?

- **Principal Security Engineer @ Trail of Bits**
- **Core developer for the Python Cryptographic Authority**
  - cryptography
  - PyNaCl
  - bcrypt
  - pyOpenSSL (ugh)
- **Simpsons lover & Frinkiac creator**
- **@reaperhulk on GitHub, Twitter, IRC**

A recent selfie

Agenda

- What is cryptography really?
- Establishing the scope
- Implementing cryptographic algorithms
- Implementing cryptographic algorithms...in Python
- Establishing what matters in your application
- FFI, Python's cryptographic super power
- Additional Mitigations
- Conclusions

What is cryptography really?

What Cryptography Is

- **In the presence of adversaries we need to be able to securely communicate information**
  - Adversaries can be passive or active
  - Communication may be synchronous or asynchronous
  - Information is not necessarily a file
- **In practice writing cryptographic software is difficult because you need both algorithms and implementation to be secure**

The Scope

Implementation-based issues caused by language-level restrictions

Implementing cryptographic algorithms

Implementing Cryptographic Algorithms

You will need...

- Low level flow control to prevent data dependent branching
- Control of caches, use of SIMD instructions, and more
- Precise memory allocation and erasure
- As much speed as you can possibly get
- An aversion to healthy life choices

Implementing Cryptographic Algorithms

Unfortunately this means it is mostly written in C. Why?

- Ubiquity
- Speed
- The memory unsafety industrial complex needs CVEs

Implementing cryptographic algorithms ...in Python!

An Illustrative Example

RSA Signing/Verification (do **not** use RSA)

- $S = M^d \pmod N$
- $M = S^e \pmod N$

RSA Encryption/Decryption (I beg of you, do not use RSA)

- $C = M^e \pmod N$
- $M = C^d \pmod N$

An Illustrative Example

RSA Signing/Verification (**do not use RSA**)

- $S = M^d \pmod N$
- $M = S^e \pmod N$

RSA Encryption/Decryption (No! Stop!)

- $C = M^e \pmod N$
- $M = C^d \pmod N$

An Aside I Can't Resist

RSA Encryption/Decryption (please do not use RSA, it is so bad)

- $C = M^e \pmod N$
- $M = C^d \pmod N$

An Illustrative Example

RSA Signing/Verification (You're going to regret it)

- $S = M^d \pmod N$
- $M = S^e \pmod N$

RSA Encryption/Decryption (Who does this? Is that you non-FS TLS?)

- $C = M^e \pmod N$
- $M = C^d \pmod N$

An Illustrative Example

A 2048-bit number

- 31617371469125627352689671147266052526164355547984376915170554077929987145352516900116515724720539689816178060234721718362346143607072885674656026837376038579878438880967403597468247996835713267788865692472164608504816442967490479790608838219997950767745655682720221478108069557673183664743508258611621304621875073764942665749430781731508992257954609822036091522236856038088605236898882200203386932640974943340571103843044257683849059728178693373593113255563469419412494705548933432997589390183592877842877427457720521348024839513788453599288317814232804104189942999307294498415740679426518847863313883638498088284383

An Illustrative Example

RSA Signing/Verification (do not use RSA and also do not do this)

- sig = pow(msg, d, n)
- msg = pow(sig, e, n)

An Illustrative Example

RSA Signin

- sig =
- msg

An Illustrative Example

RSA Signing/Verifi o this)

- sig = pow
- msg = po

An Illustrative

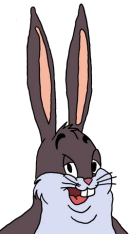
RSA Signing/Verificati is)

- sig = pow(ms
- msg = pow(s

An Illustrative

RSA Signing/Verificati is)

- sig = pow(ms
- msg = pow(s



Let's ask a scientician

Let's ask a scientician

Panic!

Establish what matters

Establish what matters

- Define the set of threats you consider in scope to protect against
- Many of these limitations may not matter for your specific situation
- There are powerful workarounds available!

FFI, Python's cryptographic super power

FFI

- Python has rich FFI to languages that speak the C ABI through cffi and ctypes so Python code can leverage native code
- This bridge gives access to every feature needed for safe cryptography, as well as allowing the construction of Pythonic APIs on top of difficult-to-use cryptographic libraries

Additional mitigations

- While byte strings are immutable bytearrays are not. You can also construct buffer protocol objects from native code to gain even more control

Secure cryptography is possible in Python

(restrictions apply)

Secure cryptography is possible in Python

- This frequently means that Python is what you use to call some underlying native code, not that the cryptographic code itself is in Python
- Threat models are critical. Building secure software requires tradeoffs, so carefully consider what you do and do not consider to be worth protecting against

References/Links

Seriously, Stop Using RSA

https://blog.trailofbits.com/2019/07/08/fuck-rsa/

The Montgomery Powering Ladder

http://cr.yp.to/bib/2003/joye-ladder.pdf

Fish in a Barrel - Memory Unsafety delenda est

https://twitter.com/lazyfishbarrel

The Black Magic of Python Wheels

https://www.youtube.com/watch?v=02aAZ8u3wEQ

Reliably Distributing Binary Modules

https://www.youtube.com/watch?v=-j4lolWgD6Q

Contact Me

Paul Kehrer

Principal Security Engineer

@reaperhulk on Twitter, GitHub, et cetera

paul.kehrer@trailofbits.com

https://langui.sh