

How to write proofs for cryptographic protocols at IETF

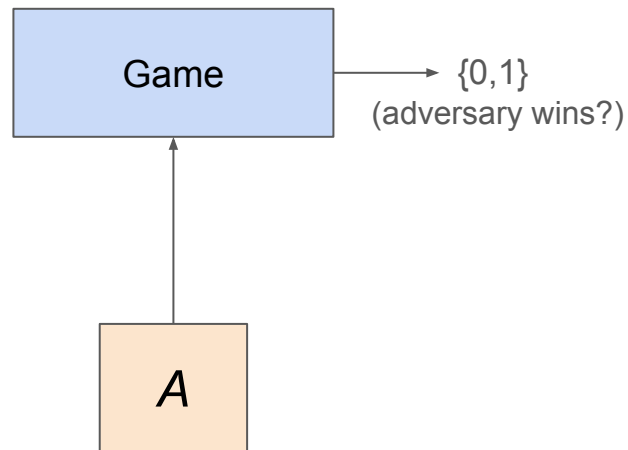
IETF 120 – UFMRG – Christopher Patton

Problem

- Many cryptographic protocols developed at IETF rely on *pen-and-paper* proofs that are easy to generate, but hard for humans to verify
 - We are doing more with cryptography than ever: higher demand for provable security and the proofs themselves are getting *longer*
 - Puts pressure on paper authors to *sketch* the proof or *punt to the appendix*, where it may never be reviewed at all
 - **Mechanization** can reduce burden on human reviewers, but requires some degree of expertise in formal methods
- Bugs in security proofs are common and sometimes belie attacks
 - OCB2, GCM, HMQV, OAEP, ...

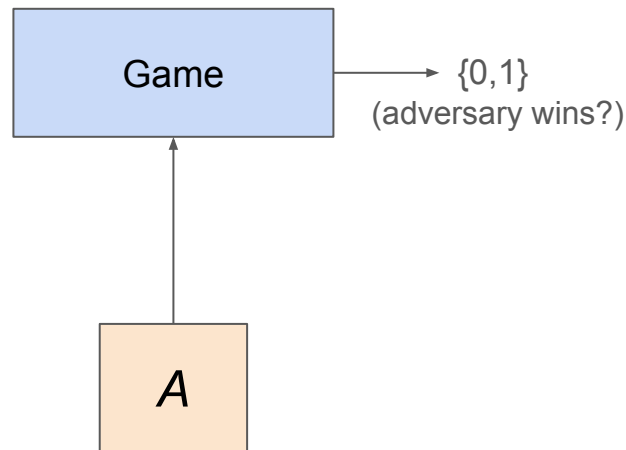
Game-playing proofs

- Define security of a cryptographic protocol via a **game** played by the **adversary**



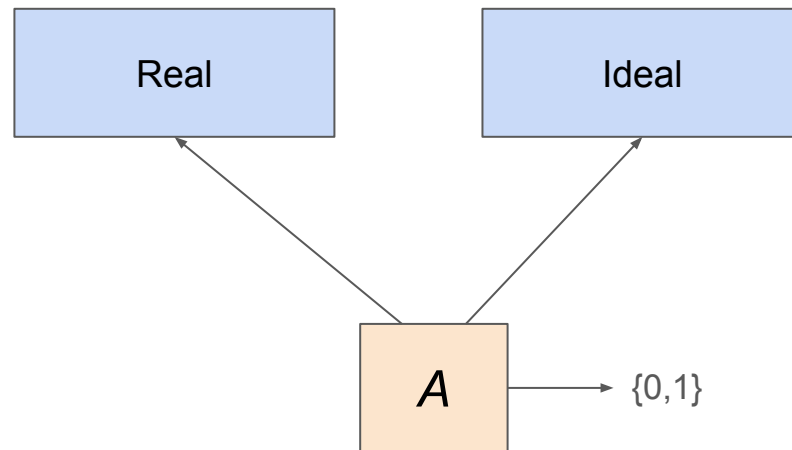
Game-playing proofs

- Define security of a cryptographic protocol via a **game** played by the **adversary**
- A **proof** relates the adversary's advantage in **winning** the game to one or more hard problems (cracking AES, solving discrete logarithms, etc.)



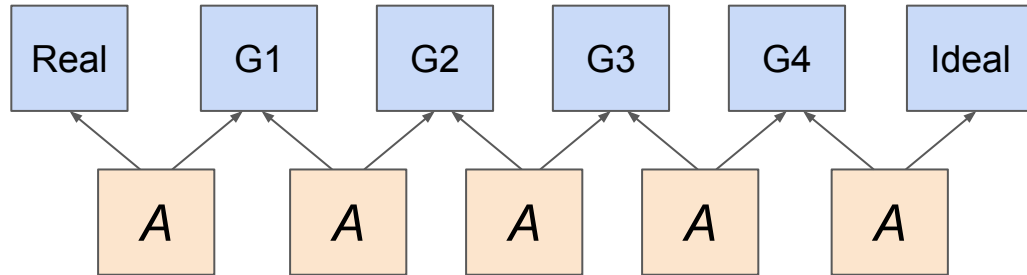
Game-playing proofs

- Define security of a cryptographic protocol via a **game** played by the **adversary**
- A **proof** relates the adversary's advantage in **winning** the game to one or more hard problems (cracking AES, solving discrete logarithms, etc.)
- Proof strategy: show that the **real** game is computationally **indistinguishable** from some **ideal** game in which the adversary has no advantage



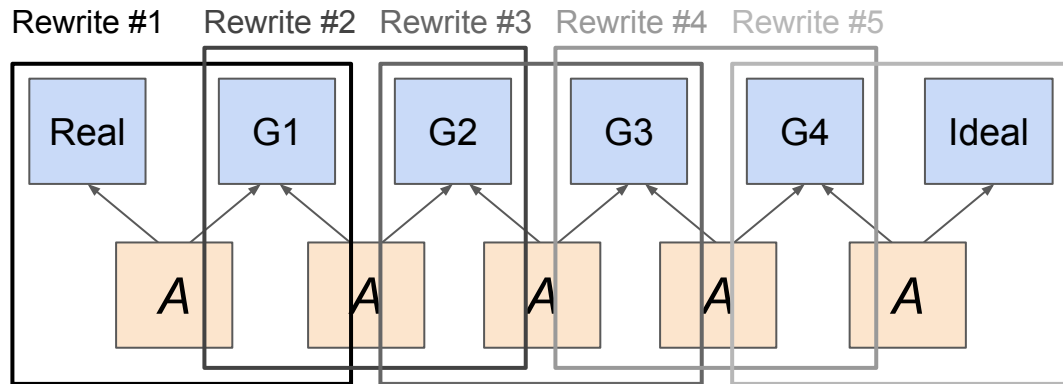
Game-playing proofs

- Proof consists of a finite **sequence** of games, starting with the real game and ending with the ideal game



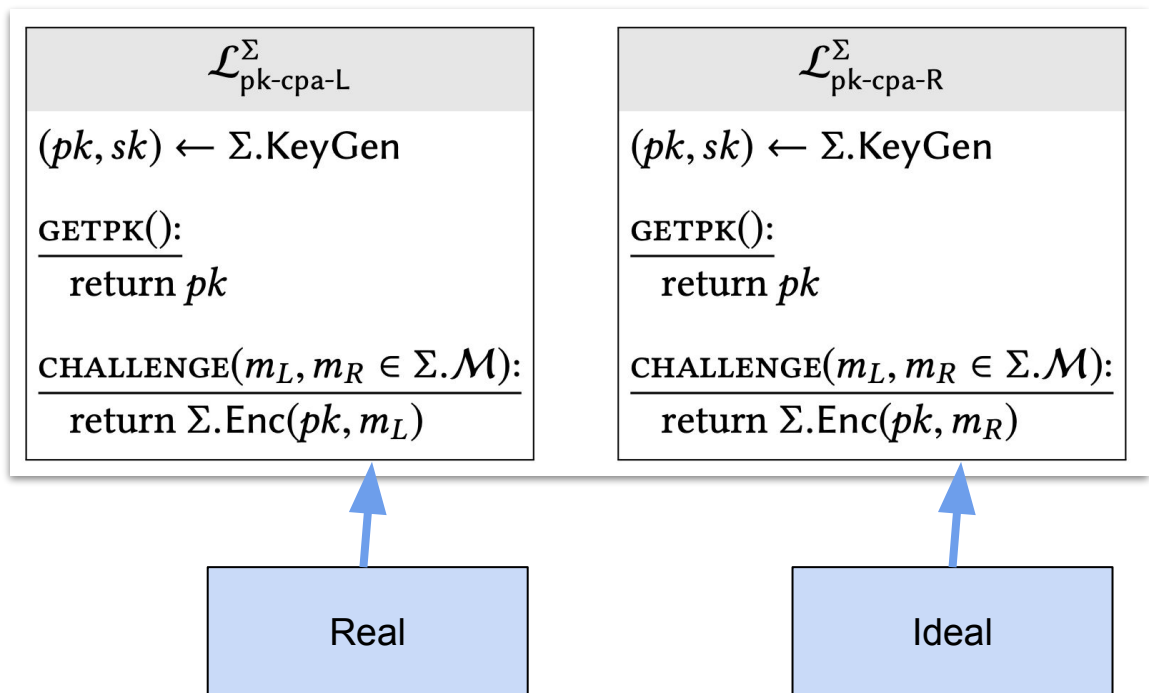
Game-playing proofs

- Proof consists of a finite **sequence** of games, starting with the real game and ending with the ideal game
- Game are specified in **code**: each game is obtained by **rewriting** the code of the previous game; the proof shows that neighboring games are indistinguishable



Example: hybrid encryption

[Definition 15.2, Joy of Crypto.](#)



- Security under chosen plaintext attack (CPA): attacker has access to a **challenge** oracle
 - Oracle encrypts either the left or the right plaintext: attacker's goal is to figure out which one

Example: hybrid encryption

- Combine symmetric encryption (**sym**) with public-key encryption (**pub**) whose message space is equal to the key space of the symmetric scheme
- generate a **temporary key** for symmetric encryption and wrap it to the recipient's public key

$$\begin{aligned}\mathcal{M} &= \Sigma_{\text{sym}}.\mathcal{M} \\ C &= \Sigma_{\text{pub}}.C \times \Sigma_{\text{sym}}.C\end{aligned}$$

KeyGen:

$$\begin{aligned}(pk, sk) &\leftarrow \Sigma_{\text{pub}}.\text{KeyGen} \\ \text{return } (pk, sk)\end{aligned}$$

Enc(pk, m):

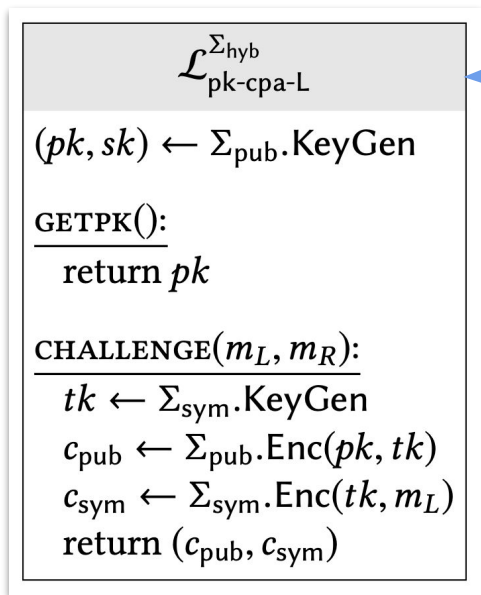
$$\begin{aligned}tk &\leftarrow \Sigma_{\text{sym}}.\text{KeyGen} \\ c_{\text{pub}} &\leftarrow \Sigma_{\text{pub}}.\text{Enc}(pk, tk) \\ c_{\text{sym}} &\leftarrow \Sigma_{\text{sym}}.\text{Enc}(tk, m) \\ \text{return } (c_{\text{pub}}, c_{\text{sym}})\end{aligned}$$

Dec(sk, (c_{pub}, c_{sym})):

$$\begin{aligned}tk &:= \Sigma_{\text{pub}}.\text{Dec}(sk, c_{\text{pub}}) \\ \text{return } \Sigma_{\text{sym}}.\text{Dec}(tk, c_{\text{sym}})\end{aligned}$$

[Construction 15.8, Joy of Crypto.](#)

Example: hybrid encryption



G1

Obtained from the Real game by modifying the challenge oracle (by inlining the call to encryption subroutine)

Indistinguishable rewrite: no semantic change, so no attacker can distinguish G1 from Real.

[Proof of Claim 15.9, Joy of Crypto.](#)

Example: hybrid encryption

G2

Obtained from G1 by generating a dummy temporary key (ciphertext now independent of the real temporary key)

Reduction: distinguisher between G1 and G2 implies CPA attacker against the pub scheme:

```
(pk, sk) ← Σpub.KeyGen

GETPK():
  return pk

CHALLENGE(mL, mR):
  tk ← Σsym.KeyGen
  tk' ← Σsym.KeyGen
  cpub ← Σpub.Enc(pk, tk')
  csym ← Σsym.Enc(tk, mL)
  return (cpub, csym)
```

[Proof of Claim 15.9, Joy of Crypto.](#)

```
CHALLENGE(mL, mR):
  tk ← Σsym.KeyGen
  tk' ← Σsym.KeyGen
  cpub ← CHALLENGE'(tk, tk')
  csym ← Σsym.Enc(tk, mL)
  return (cpub, csym)
```

◇

```

 $\mathcal{L}_{\text{pk-cpa-L}}^{\Sigma_{\text{pub}}}$ 

(pk, sk) ← Σpub.KeyGen

GETPK():
  return pk

CHALLENGE'(tkL, tkR):
  return Σpub.Enc(pk, tkL)
```

Example: hybrid encryption

G3

[not listed] Obtained from G2 by encrypting the right message instead of the right message

Reduction: distinguisher between G2 and G3 implies CPA attacker against the sym scheme:

$(pk, sk) \leftarrow \Sigma_{\text{pub}}.\text{KeyGen}$

GETPK():

return pk

CHALLENGE(m_L, m_R):

$tk' \leftarrow \Sigma_{\text{sym}}.\text{KeyGen}$

$c_{\text{pub}} \leftarrow \Sigma_{\text{pub}}.\text{Enc}(pk, tk')$

$c_{\text{sym}} \leftarrow \text{CHALLENGE}'(m_L, m_R)$

return $(c_{\text{pub}}, c_{\text{sym}})$

◇

$\mathcal{L}_{\text{ots-L}}^{\Sigma_{\text{sym}}}$

CHALLENGE'(m_L, m_R):

$tk \leftarrow \Sigma_{\text{sym}}.\text{KeyGen}$

return $\Sigma_{\text{sym}}.\text{Enc}(tk, m_L)$

[Proof of Claim 15.9, Joy of Crypto.](#)

Example: hybrid encryption

G4

[left to reader] Obtained from G3 by removing the dummy temporary key and go back to encrypting with the real one.

Reduction: distinguisher between G3 and G4 implies CPA attacker against the pub scheme.

Ideal

[left to reader] **Indistinguishable rewrite:** the Ideal game can be obtained from G4 by replacing the inlined code with a call to the hybrid scheme's encryption algorithm

Example: hybrid encryption

- This example is simple enough to verify by hand, but as the protocol or its security goal gets more complex, the proofs get longer (often 10+ pages), making it easier to make and miss mistakes
- No "standard" pseudocode in use today (put yourself in the shoes of a reviewer for a program committee with a stack of 10+ papers, each with its own pseudocode conventions)

Game $G_F^{\text{prf}}(\mathcal{A})$	Game $G_{\text{PE}}^{\text{ind}}(\mathcal{A})$	Game $G_{\text{DS}}^{\text{uf}}(\mathcal{A})$
$fk \leftarrow \mathcal{F}.\text{Keys}$	$(ek, dk) \leftarrow \text{PE}.\text{Kg}$	$(sk, vk) \leftarrow \text{DS}.\text{Kg}$
$c \leftarrow \{0, 1\}$	$b \leftarrow \{0, 1\}; C \leftarrow \emptyset$	$M \leftarrow \emptyset$
$c' \leftarrow \mathcal{A}^{\text{FN}, \text{RO}}$	$b' \leftarrow \mathcal{A}^{\text{ENC}, \text{DEC}}(ek)$	$(m, s) \leftarrow \mathcal{A}^{\text{SIG}}(vk)$
Return $(c = c')$	Return $(b = b')$	$v \leftarrow (\text{DS}.\text{Ver}(vk, m, s) = 1)$
$\text{FN}(x)$	$\text{ENC}(m_0, m_1)$	Return $(v \wedge (m \notin M))$
If $(c = 1)$ then	If $(m_0 \neq m_1)$ then return $\perp$	$\text{SIG}(m)$
$S[x] \leftarrow \text{F}^{\text{RO}}(fk, x)$	$c \leftarrow \text{PE}.\text{Enc}(ek, m_0)$	$s \leftarrow \text{DS}.\text{Sig}(sk, m)$
Else	$C \leftarrow C \cup \{c\}$	$M \leftarrow M \cup \{m\}$
If $S[x] = \perp$ then	Return c	Return s
$S[x] \leftarrow \mathcal{F}.\text{Rng}$	$\text{DEC}(c)$	
Return $S[x]$	If $(c \in C)$ then return $\perp$	
$\text{RO}(x, l)$	$m \leftarrow \text{PE}.\text{Dec}(ek, dk, c)$	
If $T[x, l] = \perp$ then	Return m	
$T[x, l] \leftarrow \{0, 1\}^l$		
Return $T[x, l]$		

Figure 1: Games for defining PRF security of a function family $\mathcal{F}$, standard security of a standard PKE scheme PE and EUF-CMA security of a signature scheme DS .

<https://ia.cr/2016/290>

<https://ia.cr/2020/573>

```

spec Key-Ind: //  $\mathcal{X}$  points to  $X_1$ ;  $\mathcal{A}$  points to  $A_2$ .
1 var  $f$  object,  $k$  int
2 var  $b, w, \text{init}, \text{guess}, \text{test bool}, s^*, i^* \text{ any}, \mathcal{S} \text{ set}$ 
3 op (SETUP):  $b \leftarrow \{0, 1\}; w, \text{init}, \text{guess}, \text{test} \leftarrow 0; \mathcal{S} \leftarrow ()$ 
4 op (1, WIN): ret  $w$ 
5 op  $\mathcal{X}^-$  (1, INIT): if  $\text{init} \neq 1$  then  $\text{init} \leftarrow 1$ ; ret  $\mathcal{X}(\text{INIT})$ 
6 op  $\mathcal{X}^-$  (1, REVEAL,  $s, i$  str):  $\mathcal{S} \leftarrow \mathcal{S} \cup \{(s, i)\}$ ; ret  $\mathcal{X}(\text{GAME ST}, \text{key}, s, i)$ 
7 op  $\mathcal{X}^-$  (1, TEST,  $s, i$  str):
8    $K_1 \leftarrow \mathcal{X}(\text{GAME ST}, \text{key}, s, i); K_0 \leftarrow \{0, 1\}^k$ 
9   if  $\text{test} \neq 1 \wedge K_1 \neq \perp$  then  $\text{test} \leftarrow 1; (s^*, i^*) \leftarrow (s, i)$ ; ret  $K_b$ 
10 op  $\mathcal{X}^-$  (1, GUESS,  $d$  bool):
11   if  $\text{guess} \neq 1$  then  $\text{guess} \leftarrow 1$ 
12    $w \leftarrow (b = d) \wedge f^{\mathcal{X}}(\mathcal{S}, (s^*, i^*)) = 1$ 
13 op  $\mathcal{A}^-$  (2,  $x$  any): ret  $\mathcal{A}(x)$ 
  
```

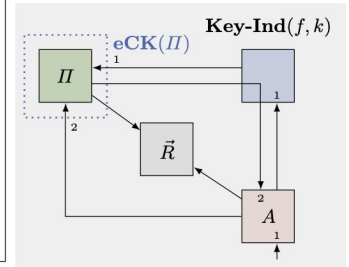
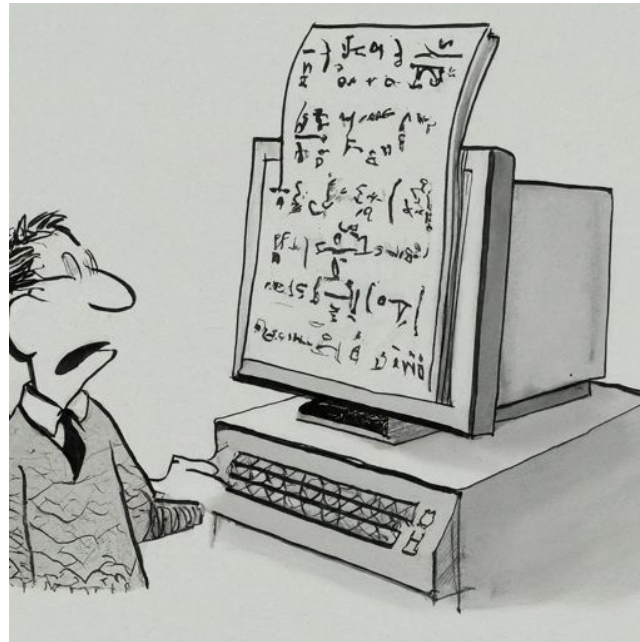


Figure 11: Left: Specification of the key-indistinguishability game $\text{Key-Ind}(f, k)$. Right: Illustration of the real experiment for adversary $\mathcal{A}$ in world $\text{Wo}(\text{Key-Ind}(f, k), \text{eCK}(\Pi))$ with resources $\vec{R}$.

Requirements

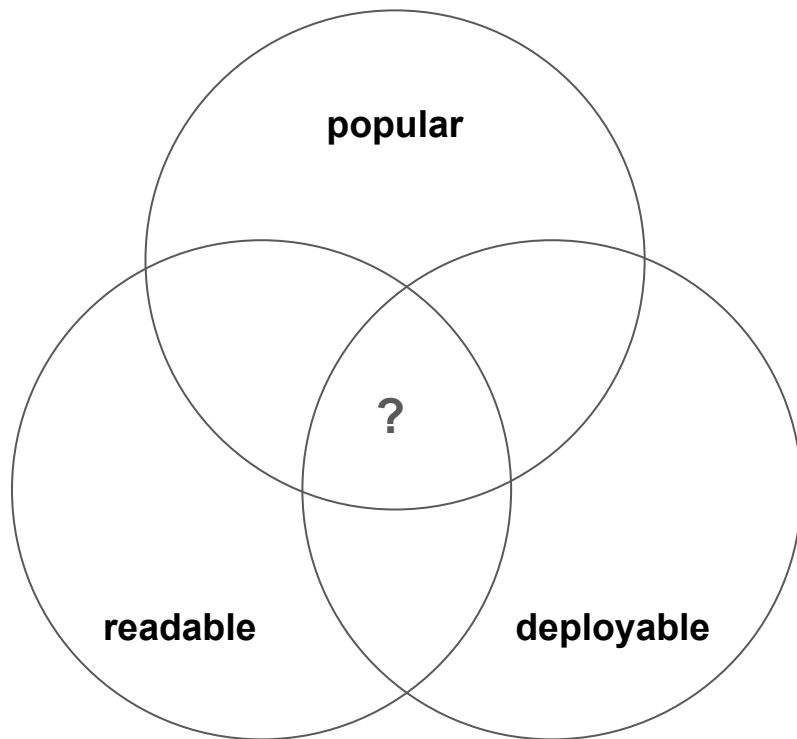
- Security proofs should be:
 - immediately amenable to mechanized verification
 - not significantly harder to generate than pen-and-paper
- Separation of concerns between:
 - **subject matter expert** that generates the proof (without knowing formal methods)
 - **formal methods expert** who verifies the proof (without domain knowledge of the application)



Google [Gemini](#)

Proofs should be *software*

- Write proofs in a programming language that is:
 - ***popular***: already in wide use at IETF
 - ***readable***: high level enough to use in specifications (no experience in the language is required in order to understand the protocol)
 - ***deployable***: used in production software



Formalizing rewrites

Claim 1 (Indistinguishable-rewrite). *For all A ,*

$$\Pr[A.\text{play}(\text{PubCpa}::\text{init}(\text{hybrid}, \text{false}))] = \Pr[A.\text{play}(\text{G1}::\text{init}(\text{hybrid}))].$$

Claim 2 (Reduction). *For all A ,*

$$\Pr[A.\text{play}(\text{G1}::\text{init}(\text{hybrid}))] = \Pr[R_{12}^A(\text{PubEnc}::\text{init}(\text{pub_enc}, \text{false}))]$$

and

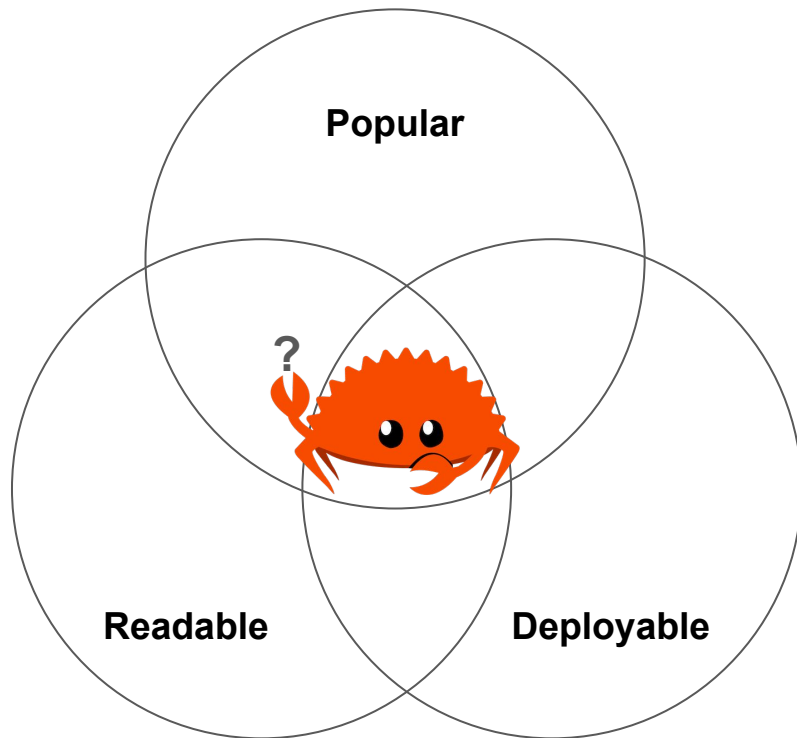
$$\Pr[A.\text{play}(\text{G2}::\text{init}(\text{hybrid}))] = \Pr[R_{12}^A(\text{PubEnc}::\text{init}(\text{pub_enc}, \text{true}))],$$

where $R_{12}^A(\text{game}) = A.\text{play}(\text{FromHybridToPubEnc}::\text{init}(\text{game}), \text{sym_enc}, \text{false})$.

<https://cjpattton.net/ietf120-ufmrg-better-proofs.pdf>

How about Rust?

- Pros:
 - Widely used at IETF and in production
 - The HAX toolchain (<https://cryspen.com/hax/>) translates a subset of Rust to F*, Coq, and ProVerif
 - Used today to produce formally verified implementations
- Cons:
 - Less readable than Python (commonly used at CFRG) but arguably more readable than Coq?



Thanks for listening!

- Example proofs to verify (*please go play with them!*): https://github.com/cjpatton/better_proofs
 - `joy::hybrid` (hybrid encryption)
 - `prf::switching` (PRP/PRF switching lemma)
 - Additional examples to come: the goal is to collect examples representative of rewrites in the literature
- More details: <https://cjpatton.net/ietf120-ufmrg-better-proofs.pdf>