What techniques do you use when writing your own cryptography methods? [closed]

Asked 16 years, 3 months ago Modified 15 years, 5 months ago

Viewed 5k times

18

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

Closed 12 years ago.

For years, maybe 10, I've been fascinated with cryptography. I read a book about XOR bit-based encryption, and have been hooked ever since thing.

I guess it's more fair to say that I'm fascinated by those who can break various encryption methods, but I digress.

To the point -- what methods do you use when writing cryptography? Is obfuscation good in cryptography?

I use two key-based XOR encryption, various hashing techniques (SHA1) on the keys, and simple things such

as reversing strings here and there, etc.

I'm interested to see what others think of and try when writing a not-so-out-of-the-box encryption method. Also -- any info on how the pros go about "breaking" various cryptography techniques would be interesting as well.

To clarify -- I have no desire to use this in any production code, or any code of mine for that matter. I'm interesting in learning how it works through toying around, not reinventing the wheel. :)

Ian

security language-agnostic encryption cryptography

Share

Improve this question

Follow

edited Jul 15, 2009 at 6:38

laalto

153k ● 67 ● 290 ● 304

asked Sep 23, 2008 at 0:17

Ian P

13k ● 6 ● 50 ● 70

See also: Lessons learned regarding Encryption and Cryptography – makerofthings7 Jun 24, 2012 at 21:04

19 Answers

Sorted by: Highest score (default)

35

The best advice I can give you is: resist the temptation to reinvent the wheel. Cryptography is harder than you think.

Get Bruce Schneier's book Applied Cryptography and read it carefully.

Share Improve this answer

Follow

edited Sep 23, 2008 at 0:24

Greg Hewgill

990k ● 191 ● 1.2k ● 1.3k

answered Sep 23, 2008 at 0:20

Tim Farley

11.9k ● 4 ● 31 ● 30

Creating your own crypto is a route to be seen in cryptogram's doghouse. E.g. schneier.com/blog/archives/2009/02/the_doghouse_ra.html – Richard Feb 22, 2009 at 22:00

I liked the "read it carefully" in the post. I had access to it once, but did not "read carefully" what I wanted. Had to come back many years later... – Dimitrios Mistriotis Aug 2, 2009 at 1:52

25

To contradict what everyone else has said so far, **go for it!** Yeah, your code might have buffer overflow vulnerabilities in it, and may be slow, buggy, etc, but you're doing this for **FUN**! I completely understand the recreational enjoyment found in playing with crypto.


That being said, cryptography isn't based on obfuscation at all (or at least shouldn't be). Good crypto will continue to work, even once Eve has slogged through your obfuscated code and completely understands what is going on. IE: Many newspapers have substitution code puzzles that readers try and break over breakfast. If they started doing things like reversing the whole string, yes, it'd be harder, but Joe Reader would still be able to break it, neve tuohtiw gnieb dlot.

Good crypto is based on problems that are assumed to be (none proven yet, AFAIK) really difficult. Examples of this include factoring primes, finding the log, or really any other NP-complete problem.

[Edit: snap, neither of those are **proven** NP-complete. They're all unproven, yet different. Hopefully you still see my point: crypto is based on one-way functions. Those are operations that are easy to do, but hard to undo. ie multiply two numbers vs find the prime factors of the product. Good catch tduehr]

More power to you for playing around with a really cool branch of mathematics, just remember that crypto is based on things that are hard, not complicated. Many crypto algorithms, once you really understand them, are mindbogglingly simple, but still work because they're based on something that is hard, not just switching letters around.

Note: With this being said, some algorithms do add in extra quirks (like string seversal) to make brute forcing

them that much more difficult. A part of me feels like I read this somewhere referencing DES, but I don't believe it... [EDIT: I was right, see 5th paragraph of this article for a reference to the permutations as useless.]

BTW: If you haven't found it before, I'd guess the TEA/XTEA/XXTEA series of algorithms would be of interest.

Share Improve this answer

Follow

edited May 23, 2017 at 11:53

Community Bot

1 • 1

answered Sep 23, 2008 at 4:17

PhirePhly

2,948 • 1 • 18 • 12

2 Nitpick: As a rule, only public key crypto is reliably based on hard-to-crack problems. Symmetric crypto and hashes are on much weaker theoretical ground. – Nick Johnson Jul 15, 2009 at 13:55

15

The correct answer is to not do something like this. The best method is to pick one of the many cryptography libraries out there for this purpose and use them in your application. Security through obscurity never works.

Pick the current top standards for cryptography algorithms as well. AES for encryption, SHA256 for hashing. Elgamal for public key.

Reading Applied Cryptography is a good idea as well. But a vast majority of the book is details of implementations that you won't need for most applications.

Edit: To expand upon the new information given in the edit. The vast majority of current cryptography involves lots of complicated mathematics. Even the block ciphers which just seem like all sorts of munging around of bits are the same.

In this case then read Applied Cryptography and then get the book Handbook of Applied Cryptography which you can download for free.

Both of these have lots of information on what goes into a cryptography algorithm. Some explanation of things like differential and linear cryptanalysis. Another resource is Citeseer which has a number of the academic papers referenced by both of those books for download.

Cryptography is a difficult field with a huge academic history to it for going anywhere. But if you have the skills it is quite rewarding as I have found it to be.

edited Sep 23, 2008 at 0:46

answered Sep 23, 2008 at 0:21

Harley Klein
182 ● 1 ● 5

9

Do the exercises here:

http://www.schneier.com/crypto-gram-9910.html#SoYouWanttobeaCryptographer

For starters, look at the cube attack paper (http://eprint.iacr.org/2008/385) and try breaking some algorithms with it. After you are familiar with breaking cryptographic schemes, you'll become better at creating them.

As far as production code goes, I'll repeat what has already been said: just use what's available in the market, since all the mainstream schemes have already gone through multiple rounds of cryptanalysis.

Share Improve this answer

Follow

answered Sep 23, 2008 at 1:04

Leo

2,890 ● 3 ● 24 ● 21

6

All the above advice is sound. Obfuscation bad. Don't put your own crypto into production without first letting the public beat on it for a while.

a couple things to add:

- Encoding is *not* encryption. I recently bypassed a website's authentication system due to the developers misunderstanding here.

- Learn how to break even the most basic systems. You'd be surprised how often knowledge of simple rotation ciphers is actually useful.
- A^B = C. You stated you've been working with two key XOR encryption. When building a cryptosystem always check that your steps are actually accomplishing something. in the two key XOR case you're really just using a different key.
- A^A = 0. XOR enryption is very weak against known or chosen plaintext attacks. If you know all or part of the plaintext, you can get all or part of the key. Plaintext ^ Cyphertext = Key
- Another good book to read is The Code Book by Simon Singh. It goes over some of the history of cryptography and methods for breaking most of the cryptosystems he covers.
- Two algorithms to learn (learn them and the history behind them):
    - 3DES: yes it's obsolete but it's a good starting point for learning fiestel and block cyphers and there are some good lessons in it's creation from DES. Also, the reasoning for the encrypt, decrypt, encrypt methodology used is a good thing to learn.
    - RSA: I'm going to display my inner math geek here. Probably the simplest encryption algorithm in use today. Methods of breaking it are known (just factor the key) but computationally

extremely difficult. m^d mod n where n = p*q (p and q prime) and gcd(d,n)=1. A little bit of group/number theory explains why this isn't easily reversed without knowing p and q. In my number theory course we proved the theory behind this at least half a dozen ways.

A note for PhirePhly:

prime factorization and discrete log are not NP-Complete, or NP-Hard for that matter. They are both unknown in complexity. I imagine you'd get a decent amount of fame from just figuring that part out. That said, the rest of your assertion is correct. Good crypto is based on things that are easy to do but hard to undo without the key.

Share Improve this answer

Follow

answered Sep 23, 2008 at 15:02

tduehr

341 ● 1 ● 3

I would go for blowfish rather than 3DES. Also AES is worth building a toy implementation of. – finnw Dec 19, 2012 at 17:17

4

Unless you're (becoming) an expert in the field, do not use home-made crypto in production products. Enough said.

Share Improve this answer

Follow

answered Sep 23, 2008 at 0:20

C. K. Young

223k ● 47 ● 390 ● 443

3

DON'T!

Even the experts have a *very* hard time knowing if they got it right. Outside of a crypto CS class, just use other people's code. Port code only if you absolutely must and then test the snot out of it with known good code.

Share Improve this answer

Follow

edited Jun 20, 2020 at 9:12

Community Bot

1 ● 1

answered Sep 23, 2008 at 0:23

BCS

78.3k ● 69 ● 194 ● 298

3

Most experts agree that openness is more valuable than obfuscation in developing cryptographic methods and algorithms.

In other words, everyone seems to be able to design a new code that everyone can break except them. The best crypto survives the test of having the algorithm and some encrypted messages put out there and having the best crypto hackers try to break it.

In general, most obfuscation methods and simple hashing (and I've done quite a few of them myself) are very easily

broken. That doesn't mean they aren't fun to experiment with and learn about.

List of Cryptography Books (from Wikipedia)

This question caught my eye because I'm currently re-reading Cryptonomicon by Neal Stephenson, which isn't a bad overview itself, even though it's a novel...

Share Improve this answer

Follow

answered Sep 23, 2008 at 0:29

CMPalmer

8,623 ● 5 ● 42 ● 50

2

To echo everyone else (for posterity), never ever implement your own crypto. Use a library.

That said, here is an article on how to implement DES:

http://scienceblogs.com/goodmath/2008/09/des_encryption_part_1_encrypti.php

Permutation and noise are crucial to many encryption algorithms. The point isn't so much to obscure things, but to add steps to the process that make brute force attacks impractical.

Also, get and read Applied Cryptography. It's a great book.

Share Improve this answer

Follow

answered Sep 23, 2008 at 0:32

noah

21.5k ● 17 ● 67 ● 88

2

Have to agree with other posters. Don't unless you are writing a paper on it and need to do some research or something.

If you think you know a lot about it go and read the Applied Cryptography book. I know a lot of math and that book still kicked my butt. You can read and analyze from his pseudo-code. The book also has a ton of references in the back to dig deeper if you want.

Crypto is one of those things that a lot of people think is very cool, but the actual math behind the concepts is beyond their grasp. I decided a long time ago that it was not worth the mental effort for me to get to that level.

If you just want to see HOW it is done (study existing implementations in code) I would suggest taking a peek at the Crypto++ library even if you don't normally code in C++ it is a good view of the topics and parts of implementing encryption.

Bruce also has a good list of resources you can get from his site.

Share Improve this answer

Follow

answered Sep 23, 2008 at 0:34

Jason Short

5,314 ● 1 ● 31 ● 44

I attended a code security session at this years Aus TechEd. When talking about the AES algorithm in .Net

2

and how it was selected, the presenter (Rocky Heckman) told us one of the techniques that had been used to break the previous encryption. Someone had managed to use a thermal imaging camera to record a cpu's heat signature whilst it was encrypting data. They were able to use this recording to ascertain what types of calculations the chip was doing and then reverse engineer the algoritm. They had way too much time on their hands, and I am fairly confident I will never be smart enough to beat people like that! :(

- Note: I sincerely hope I have relayed the story correctly, if not - the mistake is likely mine, not that of the presenter mentioned.

Share Improve this answer

Follow

answered Sep 23, 2008 at 0:36

Dr8k

1,096 ● 5 ● 11

I went to that presentation too - your story sounds like I remember it. – CAD bloke Sep 23, 2008 at 0:41

2

It's already been beaten to death that you shouldn't use home grown crypto in a product. But I've read your question and you clearly state that you're just doing it for fun. Sounds like the true geek/hacker/academic spirit to me. You know it works, you want to know why it works and try to see if you can make it work.

I completely encourage that and do the same with many programs I've written just for fun. I suggest reading this post (http://rdist.root.org/2008/09/18/dangers-of-amateur-cryptography/) over at a blog called "rootlabs". In the post are a series of links that you should find very interesting. A guy interested in math/crypto with a PhD in Computer Science and who works for Google decided to write a series of articles on programming crypto. He made several non-obvious mistakes that were pointed out by industry expert Nate Lawson.

I suggest you read it. If it doesn't encourage you to keep trying, it will no doubt still teach you something.

Best of luck!

Share Improve this answer

Follow

answered Sep 25, 2008 at 23:39

randy

356 ● 1 ● 4

1

I agree with not re-inventing the wheel.

And remember, security through obscurity is no security at all. If any part of your security mechanisms use the phrase "nobody will ever figure this out!", it's not secure. Think about AES -- the algorithm is publicly available, so *everybody* knows *exactly* how it works, and yet nobody can break it.

Share Improve this answer

Follow

answered Sep 23, 2008 at 0:28

Graeme Perrow
57.2k • 24 • 86 • 125

I have often though that the highest security code should plan for the start to publish the whole code base! The only thing that should be assumed to be secret is the keys them self. – BCS Sep 23, 2008 at 0:33

It's interesting, however, how one can make amazing indirect attacks even against something like AES: citp.princeton.edu/pub/coldboot.pdf – A. Rex Sep 23, 2008 at 0:41

I would not state that "Nobody can Break AES". I would say that (apparently) AES is currently more costly to break than the value of the information that it is typically used to protect. – Tall Jeff Sep 23, 2008 at 0:50

1

Per other answers - inventing an encryption scheme is definitely a thing for the experts and any new proposed crypto scheme really does need to be put to public scrutiny for any reasonable hope of validation and confidence in its robustness. However, implementing existing algorithms and systems is a much more practical endeavor "for fun" and all the major standards have good test vectors to help prove the correctness of your implementation.

With that said, for production solutions, existing implementations are plentiful and there should typically be no reason you would *need* to implement a system yourself.

Share Improve this answer

Follow

answered Sep 23, 2008 at 0:57

Tall Jeff

9,984 ● 7 ● 46 ● 61

1

I agree with all the answers, both "*don't write your own crypto algorithm* for production use" and "*hell yeah, go for it for your own edification*", but I am reminded of something that I believe the venerable Bruce Schneier often writes: "**it's easy for someone to create something that they themselves cannot break."**

Share Improve this answer

Follow

answered Sep 25, 2008 at 23:50

Paul Dixon

300k ● 54 ● 314 ● 349

0

The only cryptography that an non experts should be able to expect to get right is bone simple One Time Pad ciphers.

```
CipherTextArray = PlainTextArray ^ KeyArray;
```

Aside from that, anything even worth looking at (even for recreation) will need a high level degree in math.

Share Improve this answer

Follow

answered Sep 23, 2008 at 0:39

BCS

78.3k ● 69 ● 194 ● 298

1 And non-experts can screw up the OTP generation! :-P – C. K. Young Sep 23, 2008 at 0:51

From what I gather there're a lot of subtleties involved in generating cryptographically secure random numbers, and again, it's not something for non-experts to try to make production products of. :-) – C. K. Young Sep 23, 2008 at 0:52

These comments are correct. The rand() function in C or practically any other language equivalent is not cryptographically secure and can be predicted even breaking any one-time pad. Since a one-time pad has to be truly random to work this can break your system easily.
– Harley Klein Sep 23, 2008 at 1:02

Not to mention key storage, exchange, etc. Not *can* screw it up, but *will*. – AviD Sep 23, 2008 at 4:57

0

I dont want to go into depth on correct answers that have already been given (don't do it for production; simple reversal not enough; obfuscation bad; etc).

I just want to add Kerckoff's principle, "A cryptosystem should be secure even if everything about the system, except the key, is public knowledge".

While I'm at it, I'll also mention Bergofsky's Principle (quoted by Dan Brown in Digital Fortress): "If a computer tried enough keys, it was mathematically guaranteed to find the right one. A code's security was not that its pass-key was unfindable but rather that most people didn't have the time or equipment to try."
Only that's inherently not true; Dan Brown made it up.

Share Improve this answer answered Sep 23, 2008 at 5:03

Follow

AviD

13.1k • 7 • 64 • 93

0

Responding to PhirePhly and tduehr, on the complexity of factoring:

It can readily be seen that factoring is in NP and coNP. What we need to see is that the problems "given n and k, find a prime factor p of n with 1 < p <= k" and "show that no such p exists" are both in NP (the first being the decision variant of the factoring problem, the second being the decision variant of the complement).

First problem: given a candidate solution p, we can easily (i.e. in polynomial time) check whether 1 < p <= k and whether p divides n. A solution p is always shorter (in the number of bits used to represent it) than n, so factoring is in NP.

Second problem: given a complete prime factorization (p_1, ..., p_m), we can quickly check that their product is n, and that none are between 1 and k. We know that PRIMES is in P, so we can check the primality of each p_i in polynomial time. Since the smallest prime is 2, there is at most log_2(n) prime factors in any valid factorization. Each factor is smaller than n, so they use at most O(n log(n)) bits. So if n doesn't have a prime factor between 1 and k, there is a short (polynomial-size) proof which can be verified quickly (in polynomial time).

So factoring is in NP and coNP. If it was NP-complete, then NP would equal coNP, something which is often assumed to be false. One can take this as evidence that factoring is indeed not NP-complete; I'd rather just wait for a proof ;-)

Share Improve this answer

Follow

answered Feb 22, 2009 at 21:46

Jonas Kölker

7,827 ● 3 ● 45 ● 52

-1

Usually, I start by getting a Ph.D in number theory. Then I do a decade or so of research and follow that up with lots of publishing and peer review. As far as the techniques I use, they are various ones from my research and that of my peers. Occasionally, when I wake up in the middle of the night, I'll develop a new technique, implement it, find holes in it (with the help of my number theory and computer science peers) and then refine from there.

If you give a mouse an algorithm...

Share Improve this answer

Follow

answered Sep 23, 2008 at 1:48

community wiki

Jason Dufair
