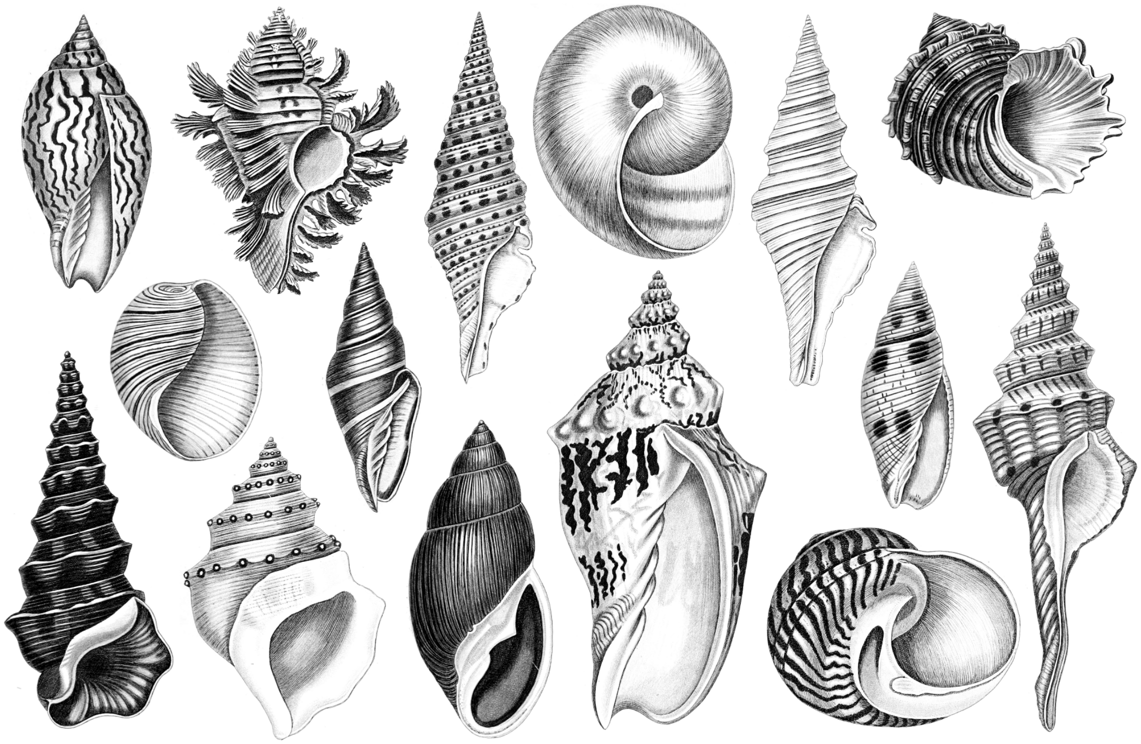

The Joy of Cryptography

Mike Rosulek ⟨mike@joyofcryptography.com⟩
School of Electrical Engineering & Computer Science
Oregon State University, Corvallis, Oregon, USA

Draft of January 3, 2021

Preface

The Joy of Cryptography is an undergraduate textbook in cryptography. This book evolved from lecture notes I developed for the cs427 course at Oregon State University (and before that, cs473 at the University of Montana).

Yes, I know that the title is ridiculous. All of the serious titles were already taken. I hope you understand that **actual joy is not guaranteed.**

What Is This Book About?

This book is about the **fundamentals of provable security.**

- **Security:** Cryptography is about controlling access to information. We break apart the nebulous concept of "security" into more specific goals: confidentiality, authenticity, integrity.
- **Provable:** We can formally define what it means to be secure, and then mathematically *prove* claims about security. One prominent theme in the book is the logic of composing building blocks together in secure ways.
- **Fundamentals:** This is an introductory book on the subject that covers the basics. After completing this course, you will have a solid theoretical foundation that you can apply to most real-world situations. You will also be equipped to study more advanced topics in cryptography.

This book is not a handbook telling you which cryptographic algorithm to use in every situation, nor a guide for securely implementing production-ready cryptographic libraries. We do not discuss specific cryptographic software (*e.g.*, PGP, Tor, Signal, TrueCrypt) or crypto*currencies* like Bitcoin. You won't learn how to become a hacker by reading this book.

Who Is This Book For?

This book is for anyone who might need to secure information with cryptography, and who is curious about what makes some things "secure" (and what makes other things insecure). I don't imagine that most readers of this book will develop their own novel cryptography (*e.g.*, designing new block ciphers), but they will be far more likely to use and combine cryptographic building blocks — thus our focus on the logic of composition.

What Background Is Needed To Understand This Book?

You will get the most out of this book if you have a solid foundation in standard undergraduate computer science material:

- Discrete mathematics (of the kind you typically find in year 2 or 3 of an undergraduate CS program) is **required background**. The book assumes that you are familiar with basic modular arithmetic, discrete probabilities, simple combinatorics, and especially proof techniques. Chapter 0 contains a brief review of some of these topics.

- Algorithms & data structures background is **highly recommended**, and theory of computation (automata, formal languages & computability) is also **recommended**. We deal with computations and algorithms at a high level of abstraction, and with mathematical rigor. Prior exposure to this style of thinking will be helpful.

Why Is Cryptography A Difficult Subject?

It's all the math, right? Cryptography has a reputation of being a difficult subject because of the amount of difficult math, but I think this assessment misses the mark. A former victim, I mean student, summed it up bluntly when he shared in class (paraphrased):

> *Some other students were thinking of taking your course but were worried that it is really math-heavy. I wouldn't say that this course is a lot of math exactly. It's somehow even worse!*

Thanks, I think.

Anyway, many corners of cryptography use math that most CS undergrads would find quite advanced (advanced factoring algorithms, elliptic curves, isogenies, even algebraic geometry), but these aren't the focus of this book. Our focus is instead on the logic of composing different building blocks together in provably secure ways. Yes, you will probably learn some new math in this book — enough to understand RSA, for example. And yes, there are plenty of "proofs." But I honestly believe you'll be fine if you did well in a standard discrete math course. I always tell my cs427 students that I'm not expecting them to love math, proofs, and theory — I only ask them to choose **not to be scared** of it.

If not math, then what? In an algorithms course, I could introduce and explain concepts with concrete examples — here's what happens step-by-step when I run mergesort on this particular array, here's what happens when I run Dijkstra's algorithm on this particular graph, here are 42 examples of a spanning tree. You could study these concrete examples, or even make your own, to develop your understanding of the general case.

Cryptography is different because our main concerns are **higher up the ladder of abstraction** than most students are comfortable with.[1] Yes, I can illustrate what happens

[1] Of course, abstraction is the heart of math. I may be making a false distinction by saying "it's not the *math*, it's the *abstraction*." But I think there's something to the distinction between a CS major's typical math-aversion and what is really challenging about cryptography.

step-by-step when you run a cryptographic algorithm on a particular input. This might help you understand **what the algorithm does**, but it can never illustrate **why the algorithm is secure**. This question of "why" is the primary focus of this book.

- Security is a **global property** about the behavior of a system *across all possible inputs.* You can't demonstrate security by example, and there's nothing to see in a particular execution of an algorithm. Security is about a higher level of abstraction.
- Most security definitions in this book are essentially: *"the thing is secure if its outputs look like random junk."* If I give an example that is concrete enough to show actual inputs and outputs, and if things are working as they should, then all the outputs will just look like meaningless garbage. Unfortunately, no one ever learned very much by staring at meaningless garbage.

Systems are *insecure* when they fail to adequately look like random junk. Occasionally they fail so spectacularly that you can actually see it by looking at concrete input and output values (as in the case of the ECB penguin). But more often, the reason for insecurity is far from obvious. For example, suppose an encryption scheme was insecure because the XOR of the first two output blocks is the same as the XOR of the third and fourth output blocks. I'm not convinced that it would be helpful to show concrete example values with this property. What's more, sometimes the reason for insecurity only "jumps off the page" on specific, non-obvious, choices of inputs.

If you want to be equipped to answer questions like "why is this thing secure but this other very similar thing is not?", then you must develop an understanding at this higher level of abstraction. You'll have to directly come to terms with abstract ideas like "this algorithm's outputs look like random junk, under these circumstances," and the consequences of these kinds of ideas. It's hard to arrive at understanding without the usual scaffolding of concrete examples (seeing algorithms executed on specific inputs), but this book is my best effort at making the path as smooth as I know how.

Known Shortcomings

- I've used this book as a primary course reference for several years now, but I still consider it to be a draft. Of course I try my best to ensure the accuracy of the content, but there are sure to be plenty of bugs, ranging in their severity. *Caveat emptor!*

  I welcome feedback of all kinds — not just on errors and typos but also on the selection, organization, and presentation of the material.
- I usually cover essentially this entire book during our 10-week quarters. There is probably not enough material to sustain an entire 16-week semester, though. I always find it easier to polish existing material than to add completely new material. Someday I hope to add more chapters (see the roadmap below), but for now you'll have to get by without some important and interesting topics.
- There is no solutions manual, and I currently have no plans to make one.

Code-Based Games Philosophy

The security definitions and proofs in these notes are presented in a style that is known to the research community as *code-based games*. I've chosen this style because I think it offers significant pedagogical benefits:

- Every security definition can be expressed in the same style, as the indistinguishability of two games. In my terminology, the games are *libraries* with a common interface/API but different internal implementations. An adversary is any calling program on that interface. These libraries use a concrete pseudocode that reduces ambiguity about an adversary's capabilities. For instance, the adversary controls arguments to subroutines that it calls and sees only the return value. The adversary cannot see any variables that are privately scoped to the library.

- A consistent framework for definitions leads to a consistent process for *proving* and *breaking* security — the two fundamental activities in cryptography.

  In these notes, *breaking* a construction always corresponds to writing a program that expects a particular interface and behaves as differently as possible in the presence of two particular implementations of the interface.

  Proving security nearly always refers to showing a sequence of libraries (called *hybrids*), each of which is indistinguishable from the previous one. Each of these hybrids is written in concrete pseudocode. By identifying what security property we wish to prove, we identify what the endpoints of this sequence must be. The steps that connect adjacent hybrids are stated in terms of syntactic rewriting rules for pseudocode, including down-to-earth steps like factoring out and inlining subroutines, changing the value of unused variables, and so on.

- Cryptography is full of conditional statements of security: *"if A is a secure thingamajig, then B is a secure doohickey."* A conventional proof of such a statement would address the contrapositive: *"given an adversary that attacks the doohickey-security of B, I can construct an attack on the thingamajig-security of A."*

  In my experience, students struggle to find the right way to transform an abstract, hypothetical B-attacking adversary into a successful A-attacking adversary. By defining security in terms of games/libraries, we can avoid this abstract challenge, and indeed avoid the context switch into the contrapositive altogether. In these notes, the thingamajig-security of A gives the student a new *constructive rewriting rule* that can be placed in his/her toolbox and used to bridge hybrids when proving the doohickey-security of B.

Code-based games were first proposed by Shoup[2] and later expanded by Bellare & Rogaway.[3] These notes adopt a simplified and unified style of games, since the goal is not to encompass every possible security definition but only the fundamental ones. The most significant difference in style is that the games in these notes have no explicit INITIALIZE

[2] Victor Shoup: *Sequences of Games: A Tool for Taming Complexity in Security Proofs.* ia.cr/2004/332

[3] Mihir Bellare & Philip Rogaway: *Code-Based Game-Playing Proofs and the Security of Triple Encryption.* ia.cr/2004/331

or FINALIZE step. As a result, all security definitions are expressed as *indistinguishability* of two games/libraries, even security definitions that are fundamentally about unforgeability. Yet, we can still reason about unforgeability properties within this framework. For instance, to say that no adversary can forge a MAC, it suffices to say that no adversary can distinguish a MAC-verification subroutine from a subroutine that always returns FALSE. An index of security definitions has been provided at the end of the book.

One instance where the approach falls short, however, is in defining collision resistance. I have not been able to define it in this framework in a way that is both easy to use and easy to interpret (and perhaps I achieved neither in the end). See Chapter 11 for my best attempt.

Other Boring Stuff

Copyright

About the cover

The cover design consists of assorted shell illustrations from *Bibliothèque conchyliologique*, published in 1846. The images are no longer under copyright, and were obtained from the Biodiversity Heritage Library (http://biodiversitylibrary.org/bibliography/11590).

Why shells? Just like a properly deployed cryptographic primitive, a properly deployed shell is the most robust line of defense for a mollusk. To an uniformed observer, a shell is just a shell. However, there are many kinds of shells, each of which provides protection against a different kind of attack. The same is true of the cryptographic building blocks we study in this course.

Acknowledgements

Some financial support for writing this book has been kindly provided by the National Science Foundation (awards #1149647, #1617197) and the Oregon State University Open Textbook Initiative.

Thanks to Brent Carmer & Leo Reyzin for many thoughtful suggestions and comments about the material. I am also grateful for the many students in cs427 who have reported countless bugs.

Changelog

2021-01-03 Chapter 2 (provable security basics) is now much more explicit about how security definitions are a "template" that we "fill in" with specific algorithms (*e.g.*, Enc, Dec). Chapter 5 (PRGs) now compares/contrasts two approaches for extending the stretch of a PRG — one secure and one insecure. This chapter also introduces a "socratic dialogue" approach to thinking about security proofs (previously there was only one such dialogue in Chapter 7). Hints to the exercises are now upside-down for extra security!

2020-02-05 Overhaul of Chapter 2 (provable security fundamentals). The structure is arguably more coherent now. The total number of examples is increased. I now also include both a successful security proof and an example of where an attempted security proof goes wrong (since the scheme is actually insecure).

2020-01-09 High-frequency winter revisions are continuing. This update focuses entirely on Chapter 13 (RSA): Many many more examples are included, *in Sage!* Discussion of CRT is (hopefully) clearer. Digital signatures content is finally there. There's a new discussion of how to actually compute modular exponentiation on huge numbers, and a couple fun new exercises.

2020-01-05 Revising in preparation for teaching CS427 during Winter term.

- Chapter 0: More examples. Expanded treatment of modular arithmetic. Tips & tricks for modular arithmetic and probabilities.
- Chapter 1: Moderate reorganization of "things that cryptographers blissfully ignore."
- Chapters 12–15: Moved AEAD chapter into position as chapter 12. Public-key stuff is now chapters 13–15.
- Chapter 13 (RSA): More (but not enough) examples of multiplicative inverses. New discussion of algorithmic aspects of exponentiation mod N. This chapter will eventually focus on signatures exclusively, but we're not year that. Expect updates over the next few months.

2019-03-21 Chapter 11 (hash functions) significant revisions: no more impenetrable security definition for collision-resistance; explicit treatment of salts; better examples for Merkle-Damgård and length-extension. New draft Chapter 15 on AEAD (after next revision will be inserted after Chapter 11).

2019-01-07 Extensive revisions; only the major ones listed here. Lots of homework problems added/updated throughout. I tried to revise the entire book in time for my Winter 2019 offering, but ran out of time.

- Added a changelog!
- Chapter 1: Kerckhoffs' Principle now discussed here (previously only mentioned for the first time in Ch 2).
- Chapter 2: Now the concepts are introduced in context of specific one-time security definition, not in the abstract. More examples of interchangeable libraries.
- Chapter 3: Polynomial interpolation now shown explicitly with LaGrange polynomials (rather than Vandermonde matrices). Full interpolation example worked out.
- Chapter 4: Better organization. Real-world contextual examples of extreme (large & small) 2^n values. Full proof of bad-event lemma. Generalized avoidance-sampling libraries.
- Chapter 5: Motivate PRGs via pseudo-OTP idea. Better illustration of PRG function, and conceptual pitfalls. How NOT to build a PRG. New section on stream cipher & symmetric ratchet.
- Chapter 6: Combined PRF & PRP chapters. Motivate PRFs via $m \mapsto (r, F(k, r) \oplus m)$ construction. Better discussion of eager vs. lazy sampling of exponentially large table. How NOT to build a PRF. New section on constructing PRG from PRF, and more clarity on security proofs with variable number of hybrids. Better illustrations & formal pseudocode for Feistel constructions.
- Chapter 7: Other ways to avoid insecurity of deterministic encryption (stateful & nonce-based). Ridiculous Socratic dialog on the security of the PRF-based encryption scheme.
- Chapter 8: Compare & contrast CTR & CBC modes.

Road Map

The following topics are shamefully missing from the book, but are planned or being considered:

1. authenticated key agreement, secure messaging / ratcheting (high priority)
2. random oracle & ideal cipher models (medium priority)
3. elliptic curves, post-quantum crypto (but I would need to learn them first)
4. DH-based socialist millionaires, PSI, PAKE, simple PIR, basic MPC concepts (low priority)

Contents