Designing Secure Cryptography

Cornell CS 6831

Notes

≪**TCR 0.1:** A couple notes for myself, will remove later.≫

- Fix a pseudocode language, including randomness, and an associated model of computation. Treat memory usage as well as run times of algorithms. We will most often treat memory as subservient to run-time, bounding the former by the latter. But special treatments are needed. The goal is to have a simplified abstraction, but which provides good predictions of running code. Probably look at "Careful with Composition" paper as starting point.

- Algorithms are simply pseudocode that take input and produce an output. Define what it means for algorithm to be runnable. Define adversaries as algorithms and require they be runnable. Comment on how this interacts with (and rules out) non-uniform reductions, other such things.

- Algorithms can be parameterized, but this is just notational sugar. The result must still be runnable. They may have oracles, and this defines some interface. Give conventions on resource usage of algorithms: number of oracle queries, run time (with oracle queries unit cost), etc. Discuss how runnable algorithms can then be composed.

- Associate probability space to any algorithm, be pedantic here since we may want to prove some basic stuff via direct manipulation of probability space (i.e., coin-counting arguments).

- Introduce security definitions as special algorithms called games. The game is just another algorithm parameterized by some adversary, scheme, etc. Thus typically what we define as a game is actually a family of algorithms, one for each instantiation of things. (How much notion of a template do we need? Is this confusing?) It is used to measure adversarial advantage, a which is a measure of success for an adversary.

- Discuss the viewpoint underlying all the above. Security models are coarse abstractions of real cryptographic systems. They must be coarse for us to do certain kinds of analysis. This means that what we prove in our formalizations do not typically apply to real systems. But they are a good heuristic: schemes for which we have good analyses tend to provide better security. We will therefore judge the value of models by their utility in helping us build cryptographic schemes that resist attackers in practice. We will include examples and discussions reinforcing this viewpoint along the way. (One example to treat right away is asymptotics. Could be good heuristic, but is often quite poor, and gives little advice about how to set security parameters meaningfully.)

- Discuss assumptions. These are formalized as games. What separates them from security definitions is contextual. They are "lower level" and less related to the goals of cryptographic protocols.

- Reductions are runnable algorithms.
- Using reductions to set security parameters. Maybe interleave some of this discussion with first couple of chapters

1 Introduction

The goal of this chapter is to describe the *perspective* taken by these notes. This perspective guides our exploration of cryptography in both overt and subtle ways, and I think it's worth dwelling on for a bit — successful scholars should necessarily be introspective about why they approach problems in certain ways, and me writing this down is such an exercise for myself.

Our focus will be on cryptography as a *tool* for securing contemporary and future information systems. Being in the digital age, this means computers and their associated communication networks. One tension we will face is how much of contemporary technology must we consider to build good cryptography, the details of which change rather rapidly, versus a more mathematical abstraction that has longer staying power. We will endeavor to strike a balance, building lasting abstractions born out of contemporary technology issus.

Classical and modern cryptography. A frequently espoused viewpoint is that one can divide between modern cryptography and classical cryptography. The tipping point being perhaps the seminal work of Shannon in the 1940s [**shannon1949communication**], or that of Goldwasser and Micali [**shafi1984probabilistic**] (and their contemporaries) or Dolev and Yao [**dolev1983security**] in the 1980s. In either case the idea is that these works introduced a key new element into cryptography: proofs of security. Previous to these works, in classical cryptography, one built cryptographic algorithms and protocols and informal arguments of their security, for some often vague notion of security. Indeed early works talk about the inability to recover a message from a ciphertext without the decryption key, without really specifying what are the characteristics of these messages, how keys are chosen, and more.

Shannon in his work provided a beautiful mathematical definition of security for symmetric (or secret key) encryption, called perfect secrecy. Roughly it states that given a ciphertext, the probability, taken over the choice of secret key, that it is an encryption of any message is the same. It's simple to state and intuitive, and provides seemingly the best possible security, hence the name. In the 1980s two different viewpoints emerged. On one hand, following Goldwasser and Micali, arose the view that we should take into account computational complexity. Roughly stated for Shannon's symmetric encryption setting, we should only care that no computationally efficient adversary can learn something about messages given ciphertexts. Computational efficiency was measured in asymptotic terms, though Bellare and Rogaway pioneered a concrete security approach that enabled a more granular accounting of computational resources in a sequence of papers in the 1990s [**bellare1994security**]. This line of work built off techniques from computational complexity theory, including the use of manually written reductions relating difficulty of an adversary breaking a scheme to the difficulty of breaking some underlying believed-to-be computationally hard problem. In parallel a body of work built off Dolev and Yao's ideas, in which proofs that dispensed with computational complexity, primarily treated lower level primitives as "perfect" symbolic operations, and utilized techniques from the theory of programming languages. It is of historical and sociological interest that the two threads of research built two largely disjoint academic communities (researchers, conference venues, etc.).

Taken together, the use of precise security goals stated in mathematical language and frameworks for proving schemes protocols secure represents a big shift in the way cryptography has been analyzed. In the last 40 years especially, we have seen a blossoming of public work in cryptography focused on proving security, with a vibrant academic community focusing on it.

The ground truth of (in)security. Security is inherently normative and context-dependent. We will take a pragmatic viewpoint on evaluating security: Are real attacks being foiled? By real attacks, I mean evidenced cases in which actors have sought to, or more likely, achieved harm against some computing system. We will have to use our normative judgement on what is harm, but the point here is that, for the researcher, we will attempt to steer our design and evaluation of cryptography towards practical security issues. Cryptosystem A will dominate cryposystem B if we can point to real attacks that it foils while cryptosystem B only provides some hypothetical improvements. Evidence of real attacks can be garnered through word-of-mouth discussions with practitioners, or even research into attackers motives and means.

Unfortunately judging efficacy is not yet formulaic, and coming up with rigorous ways to predict whether cryptographic designs prevent attacks represents an opportunity area for much future research.

Methodoligical regimes. Towards better predictions, we want principled approaches. One can roughly group together various methodological regimes cryptographers use to evaluate whether a cryptographic design will resist attack. Examples include:

- Cryptanalysis (here I mean the study of blockciphers, hash functions, number-theoretic primitives, and other low-level cryptographic primitives)
- Developing attacks (against algorithms and protocols, or implementations of them)
- Algorithm verification (such as Dolev-Yao type analyses) or even software verification
- Reduction-based analysis (very often called provable security)
- Empiricism (understanding cryptographic system behavior in the real world)
- User studies (to determine if cryptographic systems easy implement and use securely)

Each regime offers instruments for scrutiny of cryptography, and they each have their strengths and weaknesses. If one wants to understand how attackers can exploit an implementation of a cryptosystem, it's probably not best to start with verification or reduction-based security — start by playing the role of an attacker and applying your experience and ingenuity. On the other hand, if you want to rule out attacks that violate the lower-level structure of the cryptographic algorithms, then using reduction-based analysis to show that the design appears to be secure as long as some underlying hard problem remains so. If you want to understand if problems are hard, you're back to the body of cryptanalytic work attempting to build fast algorithms for factoring, discrete log, or finding collisions in hash functions. If you have an interactive protocol for which manual analysis exceeds typical expert human capacity, then you should probably start looking towards symbolic analysis tools.

In line with our utilitarian viewpoint, you picked the right tool not because of success measured by aesthetic concerns such as the cleverness of a demonstrated attack, the elegance of a reduction, or improvements in type theory, but rather because it helped you build a system resisting the types of attacks seen in practice. To really gain confidence should most typically use a combination of methodological regimes to gain confidence in a design, such as was used in a large effort for the recent TLS 1.3 specification, which used a combination of almost all the regimes above. A sophisticated cryptographer will of course, appreciate and try to understand the gaps between different methodological regimes – the attacks that they don't, in aggregate, rule out.

Unfortunately, I don't know much about approaches — there are only so many hours in one's life — and so many will sadly only get cursory discussion. I will focus a lot on reduction-based analysis in the concrete-security tradition, developing attacks, and empiricism, the topics for which I have some experience. They will serve as good examples to understand the rough boundaries between different methodological regimes and the fascinating issues that arises at those boundaries.

The power and pitfalls of abstraction. A key element to any security analysis is abstraction, and security analyses use abstraction aggressively to ensure analytical tractability. Abstractions in our context is an exercise in *modeling* of cryptographic settings. For example, we will spend a lot of time developing security games that capture desirable security properties for cryptographic algorithms.

Modeling is hard. The reason is we must simultaneously balance the need for simplicity of an abstraction with the danger of omitting security-critical details. As a ready example, almost all of our modeling will not countenance side-channel attacks such as those emanating from analyzing power traces or microarchitectural nuances. Would we be better off with models that do try to take these into account? In most cases the seeming answer is 'no', because the models would become too complicated to allow useful analysis.

We therefore advocate an iterative approach to modeling that roughly goes as follows:

(1) Identify normative security concerns. Or put more simply: what attacks do we believe people care about preventing?

(2) Develop one or more cryptographic security models that (hopefully) speak to these attacks. Analyze different models formally by comparing them. Develop cryptographic systems that we can show analytically are secure in the model.

(3) Return to normative security goals. Can we break our proposed schemes in some normatively damaging way?

I use the term normative here to indicate the fact that, ultimately, security is a social construct and we are driven by people's desires. We won't, in this class, put a significant amount of attention on tools for wading through the ethical and sociological questions of how we should guide our normative concerns — that requires philosophy.[1] That said, we can identify interesting security concerns via intuition, experience, and keeping tabs on socially important topics. My favorite approach is to pay close attention to what problems practitioners are facing — what attacks have arisen in practice that cryptography might solve? Develping the ability to identify such opportunities can be a good way to build a career of impactful research.

Once normative concerns are identified, we can attend to how we can address them as rigorously as possible. Here we come up with security models, often called security definitions. We will build off the tradition of what's called provable security cryptography that provides definitions in the way of probabilistic experiments. We will discuss how to formalize in subsequent chapters. Within these abstractions we will also be able to define absractions of cryptographic tools: these are the pseudocode versions of real software or hardware. These serve as a specification of sorts, though often these are, like security models, woefully lacking in details from the perspective of an implementor. A key issue will be attending to this tension between level of detail and modeling simplicity.

We will then show how to do various formal analyses in these abstract models. The approach will be hand-written proofs, in a mathematical style. They will most often be reduction-based and we will focus on ensuring, whenever possible, that analyses result in actionable information about the security of a proposed cryptographic algorithm.

We will then return to understanding limitations of what we've produced by way of developing attacks against normative security goals. Often we can iteratively develop formal models to attend to these attacks, helping suss out subtleties missed in earlier iterations. However, at some point we expect that more modeling will have diminishing returns: either the normative attacks can't be found or seem more easily dispensed with via informal guidance (e.g., use padding when necessary to obscure lengths, use constant-time code).

[1] One might read Rogaway's paper on this topic for further pointers `http://web.cs.ucdavis.edu/~rogaway/papers/moral-fn.pdf`.

Given our utilitarian viewpoint, we will want, throughout, to attend to practicality: Are the cryptographic algorithms we arrive at reasonable for use? Are they fast enough for practice? Will they be fragile to implement? If the answers are no, that doesn't necessarily mean that some effort should be discarded — it may be fascinating theory that has value for the future. But we should at least be able to articulate how far from practical some proposal is, and why. This often requires experimentation, measurement, and dealing with often messy issues such as legacy requirements or psychological acceptability by developers.

The viewpoints above may stand in stark contrast to the rhetoric of provable security more often used in modern treatments of cryptography. What people refer to as provable security is just the step (2) above, but I try to avoid this terminology because I think (and experience indicates) that it oversells and may often under-deliver. Proofs can have errors. Models can become so complex they have limited analytical utility and using them leads to mistakes or overly complex protocols that are hard to implement. In any case, while for theoreticians provable security is a meaningful term, for cryptography meant to be secure in practice, it risks being misleading. Nothing is so black-and-white in practice.

This is not to say that theory, modeling, and formal analysis aren't critical to applied cryptography. Most obviously we will be building off the deep literature from theoretical cryptography, and without that theory we would be lost. More directly, theory provides us powerful tools to help us rule out large classes of attacks for deployed schemes, that complement and strengthen the one-at-a-time approach of simple attack-driven investigations. One can visualize this process for a particular scheme as shown in Figure **??**, which means to depict the universe of all possible attacks and those attacks covered by different analysis approaches. Each point in this universe represents an implementable attack against an instantiation of the system, which may or may not succeed, and the goal of the analyst is to rule out the efficacy of as many such attacks as possible. The black dots represent one-at-a-time investigation ruling out individual attacks, the green circle might represent formal reduction-based analyses, the blue circle symbolic analyses, and the red circle an analysis of some class of side-channel attacks. Of course this is highly abstracted, but gives a sense of the thinking: formal modeling and analysis can help rule out larger classes of attacks, but certainly will miss things that fall outside the scope of the model. By combining approaches in a thoughtful manner combined with normative intuition, one achieves better coverage and can spot attack strategies that may be more likely to work.

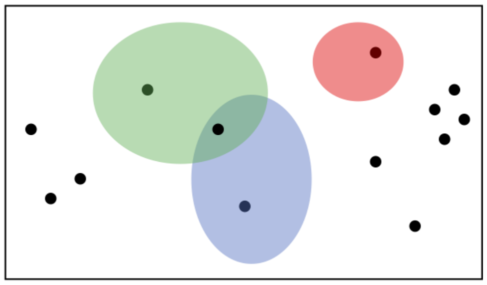
Figure 1: The universe of possible attacks against a cryptographic scheme or protocol.

To make this kind of approach work, though, we must have a mastery of modeling and formal analysis tools. Gaining that mastery is the subject of this course.

2 Preliminaries and Notation

We collect here some notation that we will use throughout these notes. ≪**TCR 2.1:** If you introduce new notation in writing up a lecture, please consider adding it here.≫

Basics. We most often denote sets by calligraphic, capital letters, such as $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$. A discrete probability distribution is a set Ω, the event space, together with a function $p \colon \Omega \to [0,1]$ for which $\sum_{\omega \in \Omega} p_\Omega(\omega) = 1$. We write p_Ω when we need to disambiguate the event space, but otherwise simply p when Ω is clear from context. We will also, following convention, often write $\Pr[\omega]$ instead of $p(\omega)$. The support of p is defined to be the set $\mathrm{Supp}(p) = \{\omega \mid \omega \in \Omega \wedge p(\omega) > 0\}$, i.e., the set of all points in Ω that have non-zero probability. As a slight abuse of notation, we can write $\Pr[\mathcal{S}] = p(\mathcal{S}) = \sum_{\omega \in \mathcal{S}} p(\omega)$ for any set $\mathcal{S} \subseteq \Omega$.

A random variable X is a map $Y : \Omega \to \mathcal{X}$ from some event space Ω with associated probability distribution p_Ω over a set Ω. So for some other set $\mathcal{S}$ we let $\Pr[X \in \mathcal{S}] = \Pr[\{\omega \mid X(\omega) \in \mathcal{S}\}]$ to be the probability that the random variable takes on a value in $\mathcal{S}$, over the probability distribution p_Ω.

We use the language of probability as the foundation for formalizing cryptographic algorithms, security, and more. Interestingly the probability spaces involved get complicated quickly, and a common problem is that they end ambiguous. We will therefore rely on a crutch that has proved quite useful for communicating, and reasoning about, probability distributions.

Pseudocode games. We fix some pseudocode language to describe security models, cryptographic algorithms, and more. Roughly we will follow the notational tradition established by Bellare and Rogaway in the 2000s [**bellare2006security**], but using slightly different syntax/symantecs that are based most closely on a treatment from [**ristenpart2011careful**]. Code-based games are convenient for more precisely defining probability spaces, which are what we use to model security and correctness goals, algorithms, and more.

We will use procedures, variables, and typical programming statements (such as operators, loops, procedure calls, etc.). Typical statements are shown in Figure **??**. We do not provide a formal specification of the programming language, see [**bellare2006security**] for an example of doing so. We will rely on some conventions to help make sense of games. Types should be understable from context. The names of syntactic objects (procedures, variables, etc.) must be distinct. Variables are implicitly initialized to default values: integer variables are set to 0, arrays are everywhere $\bot$, etc. Here $\bot$ is a distinguished symbol by tradition used to denote an error in the cryptographic literature. By distinguished we mean that it assumed to not be used for any other reason, not appearing in alphabets over which strings are taken, etc.

A *procedure* is a sequence of statements together with zero or more variable inputs and zero or more outputs. An *unspecified procedure* is a procedure whose pseudocode, inputs, and outputs are understood from context. We will see some examples of unspecified procedures, the most frequent in our security games being the *adversary* which is often left unspecified. A call to a procedure requires providing it with inputs, running its sequence of statements, and returning its output. We will interchangeably use the term call and *query* for procedure invocation. A procedure P can itself query other procedures. The set of procedures $Q_1, \ldots, Q_k$ called by a procedure are statically fixed, and we require that there are no type mismatches in inputs

$x \leftarrow y$	Assignment
$x \leftarrow\$\ \mathcal{X}$	Uniform sampling from a set
$z \leftarrow\$\ P(x,y)$	Call a randomized procedure
$z \leftarrow P(x,y)$	Call a deterministic procedure
Ret x	Return from a procedure
$z \leftarrow x \parallel y$	String concatenation
$(x,y) \xleftarrow{n} z$	Parse string z s.t. $\lvert x \rvert = n$

Figure 2: Some statements used in our games.

and outputs.

Say that the code of P expects to be able to call k distinct procedures. We will write $P^{Q_1,Q_2,\ldots,Q_k}$ to denote that these calls are handled by $Q_1, Q_2, \ldots, Q_k$ and implicitly assume (for all $i \in [k]$) that there are no syntactic mismatches between the calls that P makes to Q_i and the inputs of Q_i, as well as between the return values of Q_i and the return values expected by P.

We assume that all procedures eventually halt, regardless of randomness used, returning their outputs, at which point execution returns to the calling procedure. Two procedures P_1 and P_2 are said to *export the same interface* if their inputs and outputs have the same number and types.

Variables will be local by default, meaning they can only be used within a single procedure. Variables are static, meaning that they retain their state between calls to the procedure. It will be convenient to allow sharing of variables at times, for which we use a *collection of procedures*. This is a set of one or more procedures whose variables have scope covering all of the procedures. We will denote a collection of procedures using a common prefix ending with a period, so for example $(P.x, P.y, \ldots)$, Sometimes we will use the term interfaces for the specific prefixes of the a collection of procedures P.

A *main procedure* is a special procedure that takes no inputs and has some output. We will mark it by **main** (though below we'll see some syntactic sugar that provides greater brevity). No procedure may call **main**, it can access all the variables of other specified procedures (though not unspecified procedures).

A game consists of a main procedure together with a set of zero or more specified procedures. We write G for a game. A game may also make use of unspecified procedures (such as adversaries), which we enumerate as superscripts, e.g., $\mathrm{G}^{P_1,P_2,\ldots,P_k}$. In most games used as security definitions, one (or more) of the unspecified procedures will be called the adversary, most often denoted by $\mathcal{A}$. For a given instantiation of the unspecified procedures, one can run a game: execute its statements starting with the designated **main** procedure, and ultimately outputing whatever **main** returns.

Run times and random variables. Games can make random choices, due to the supported statements for sampling according to a distribution. We can associate to games a model of computation, which specifies how much running each (type of) statement costs in terms of time, memory, or both. A typical model is to assign to each statement the same abstract unit cost, and the run time then becomes the number of statements executed in the course of the game. When procedures are called, we attribute the unit cost of the call statement to the caller and the remaining cost of executing the procedure's statements to the callee.

By default we will require that games terminate in some finite number of steps t, and clarify explicitly when this does not hold. The number of queries made by the main procedure, or any other procedure for that matter including unspecified ones, is therefore also upper bound by t. We may often limit the number of queries made by an adversary to some maximum number $q \leq t$.

Given these finiteness conditions, we similarly know that there is a finite limit on the number of random samples made in a game. Since we restricted to random sampling from finite sets, we have that for any game G there is an event space Ω_{G} and an associated probability distribution defining the output of the game G. Given our restrictions, Ω_{G} is a set of possible values, the cross-product of all the random sampling procedures within G. We sometimes refer to Ω_{G} as the *coins* of the game. For some fixed unspecified procedures $P_1, \ldots, P_k$ we denote the event that executing the game $\mathrm{G}^{P_1,\ldots,P_k}$ outputs a particular value y by "$\mathrm{G}^{P_1,\ldots,P_k} \Rightarrow y$" and the associated probability over Ω_{G} is denoted $\Pr[\mathrm{G}^{P_1,\ldots,P_k} \Rightarrow y]$. When y is clear from context we will omit it, writing instead $\Pr[\mathrm{G}^{P_1,\ldots,P_k}]$. For example, we will often have games output a boolean and then y will most often be the value true.

We can similarly associate to any variable within a game an event within Ω_{G}. These can also be equivalently considered to be random variables on domain Ω_G. Our convention will be to overload notation, and define the event that a variable X in a game G takes on a certain value y as simply "$X = y$" with associated probability $\Pr[X = y]$ over the coins of Ω_{G}.

Runnable games. We want to emphasize a point, which is that games are by our conventions above runnable. That means that, once any unspecified procedures are fixed, you could write a program in a conventional programming language, and actually run the game on a real computer. Obviously like with all abstract algorithms, the actual run times will vary, but assuming relative efficiency the game will complete.

It relatively frequently arises in proofs that one deviates from runnable games. A common example is logic of the following form. Let $\mathcal{A}$ be an adversary, and consider a game $\mathrm{G}^{\mathcal{A}}$. Remember we implicitly assume that $\mathcal{A}$ is compatible with G, and we have that $\mathrm{G}^{\mathcal{A}}$ is runnable. Now consider the set of all adversaries compatible with $\mathcal{A}$, and let $\mathcal{A}_{\max}$ be the member of this set that maximizes $\Pr[\mathrm{G}^{\mathcal{A}} \Rightarrow \mathsf{true}]$. But now $\mathrm{G}^{\mathcal{A}_{\max}}$ is no longer runnable, because $\mathcal{A}_{\max}$ is not concretely specified. While mathematically $\mathcal{A}_{\max}$ is well-defined, there is no clear way to write down its code, even given the code defining $\mathcal{A}$. We will try whenever possible to avoid such arguments, as they have various subtle implications that are, in general, not great for making clear claims about security.

3 Ciphers and Initial Security Notions

Ciphers. We start by defining a cipher. A cipher $\mathcal{E} = (E, D)$ is defined by a a pair of deterministic algorithms E and D. To any cipher $\mathcal{E}$ we associate sets called the key space $\mathcal{K}$, message space $\mathcal{M}$, and ciphertext space $\mathcal{C}$. We do not surface these sets in the notation for a cipher, and we will require that the association be clear from context.

Each algorithm has two inputs. Enciphering takes a key $K \in \mathcal{K}$ and message $M \in \mathcal{M}$, and outputs a ciphertext $C \in \mathcal{C}$. Because E is deterministic, we can equally formalize it as a map $E\colon \mathcal{K} \times \mathcal{M} \to \mathcal{C}$. For a given key K we let $E_K : \mathcal{M} \to \mathcal{C}$ be defined by $E_K(M) = E(K, M)$ for all $M \in \mathcal{M}$. Deciphering takes a key $K \in \mathcal{K}$ and ciphertext $C \in \mathcal{C}$ and outputs a message $M \in \mathcal{M}$. Again, we can view it as a map $D\colon \mathcal{K} \times \mathcal{C} \to \mathcal{M}$.

Both E and D must be efficiently computable for all $K \in \mathcal{K}$. (We have not defined efficiently computable and use the term here informally.) We require that a cipher be correct, meaning that $\forall M \in \mathcal{M}, \forall K \in \mathcal{K}, D_K(E_K(M)) = M$.

Example 1 One simple example of a cipher is the **one-time pad** (OTP). Let $\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0,1\}^n$ for some $n \in \mathbb{N}$. Then we define OTP as follows:

$$E_K(M) = M \oplus K$$
$$D_K(C) = C \oplus K$$

Claude Shannon proved that OTP is perfectly secure in 1949 [**shannon1949communication**].

Security notions. What do we intuitively expect of a cipher? Minimally:

- The secret key should remain secret.
- The message should remain secret.

Let's try to formalize these notions. We will start with a security notion called **target key recovery security** (TKR). As the name suggests, the goal of the adversary is to recover the challenge key given a chosen plaintext attack, meaning the adversary can choose which messages to query to the Fn oracle. The game pseudocode is provided in Figure **??**. We let TKR$_{\mathcal{E}}$-advantage of a TKR$_{\mathcal{E}}$-adversary $\mathcal{A}$ be defined by

$$\mathbf{Adv}^{\mathrm{tkr}}_{\mathcal{E}}(\mathcal{A}) = \Pr\left[\, \mathrm{TKR}^{\mathcal{A}}_{E} \Rightarrow \mathsf{true} \,\right] .$$

We must now ask ourselves how well TKR captures the security of a cipher. Let us first try to analyze the security of OTP using this definition. Notice that OTP actually fails to provide TKR security, which we show with the following adversary.

```
adversary $\mathcal{A}$
$K \leftarrow \mathrm{Fn}(0^n)$
Return $K$
```

$\mathcal{A}$ simply queries for 0^n, which returns $0^n \oplus K = K$, thereby recovering the challenge key with $\mathbf{Adv}^{\mathrm{tkr}}_{\mathcal{E}}(\mathcal{A}) = 1$. This then means that OTP is actually insecure according to the TKR security definition. But as we noted earlier, OTP is considered secure! Our definition then fails to capture the security of OTP.

Now consider the identity cipher $E_K(M) = M$ for $\mathcal{K} = \{0,1\}^k$. Since the cipher simply returns the message, no information about the key is included in the ciphertext. The best an adversary can do is return a random key, which has probability 2^{-k} of being the correct target key. Then for any adversary $\mathcal{A}$, it holds that $\mathbf{Adv}^{\mathrm{tkr}}_{\mathcal{E}}(\mathcal{A}) = 2^{-k}$, meaning the identity cipher is secure. Clearly this is not the case, and thus TKR security does not provide message confidentiality. Furthermore, this security notion is "unfair" to an adversary, since there can be many keys that are *consistent* on a query transcript.

We then look at a different notion called **key recovery security** (KR). Under this definition, if an adversary outputs a key that is consistent with the query transcript, then it wins. The game pseudocode is provided in Figure **??**. We let $\mathrm{KR}_{\mathcal{E}}$-advantage of a $\mathrm{KR}_{\mathcal{E}}$-adversary $\mathcal{A}$ be defined by

$$\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{E}}(\mathcal{A}) = \Pr\left[\,\mathrm{KR}^{\mathcal{A}}_{E} \Rightarrow \mathsf{true}\,\right]\ .$$

How does KR compare to TKR? We now look at how to formally compare security definitions to gain an understanding of the relationship between TKR and KR.

Comparing security definitions. To show that some definition DEF1 does not imply another definition DEF2, we can show a *counter-example.* This requires producing a scheme such that we can show that no (reasonable) DEF1-adversary has a good advantage. We then give a DEF2-adversary that does maintain a good DEF2 advantage.

Conversely, to show that DEF1 does imply DEF2, we can show a *reduction.* This requires converting a DEF2-adversary $\mathcal{A}$ into a DEF1-adversary $\mathcal{B}$ such that $\mathcal{B}$'s DEF1 advantage upper bounds $\mathcal{A}$'s DEF2 advantage.

Example 2 TKR $\not\Rightarrow$ KR
To show this, we need a counter-example, and in this case we can use the identity cipher $E_K(M) = M$ for $\mathcal{K} = \{0,1\}^k$ and $\mathcal{M} = \{0,1\}^n$. We have already discussed that any adversary cannot get a TKR advantage greater than 2^{-k}. Next we must provide a KR-adversary that achieves a "good" advantage. We construct a KR-adversary $\mathcal{A}_{KR}$ that simply returns 0^k. Since $\forall M \in \mathcal{M}, \forall K, K^* \in \mathcal{K}, E_K(M) = E_{K^*}(M) = M$, $\mathcal{A}_{KR}$ achieves advantage 1. Thus, we have shown that TKR $\not\Rightarrow$ KR. ▮

We now ask whether KR $\Rightarrow$ TKR and prove this with the following theorem.

Theorem 1 *Let $\mathcal{E}$ be a cipher. For any $\mathrm{TKR}_{\mathcal{E}}$-adversary $\mathcal{A}$, we give a $\mathrm{KR}_{\mathcal{E}}$-adversary $\mathcal{B}$ such that* $\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{E}}(\mathcal{A}) = \mathbf{Adv}^{\mathrm{tkr}}_{\mathcal{E}}(\mathcal{B})$.

Proof of Theorem ??: We are given an adversary $\mathcal{A}$ that wins the $\mathrm{TKR}_{\mathcal{E}}$ game with advantage $\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{E}}(\mathcal{A})$, meaning that $\mathcal{A}$ will return the target key chosen by the game with probability $\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{E}}(\mathcal{A})$. We now want to construct an adversary $\mathcal{B}$ that wins the $\mathrm{KR}_{\mathcal{E}}$ game and do so with the following.

```
adversary $\mathcal{B}^{\mathrm{Fn}}$
$K^* \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Fn}}$
Return $K^*$
```

$\mathcal{B}$ is given access to its oracle Fn, and it runs $\mathcal{A}$ using this same oracle. Notice that $\mathcal{B}$ can simply provide $\mathcal{A}$ its own oracle because the distribution of outputs for oracle Fn in game $\mathrm{TKR}_{\mathcal{E}}$ and for oracle Fn in game $\mathrm{KR}_{\mathcal{E}}$ are equivalent. This is known as an *elementary wrapper*: a reduction that runs an adversary and simulates an oracle in the simplest way possible.

Now $\mathcal{A}$ returns the precise target key chosen by the game, so $\mathcal{B}$ returns this key because the target key will always be consistent with all queries. $\mathbf{Adv}^{\mathrm{tkr}}_{\mathcal{E}}(\mathcal{B})$ is then the probability that the key returned by $\mathcal{A}$ is the target key, which is $\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{E}}(\mathcal{A})$. ▮

In our theorem statements including reductions, we need to interpret the words "we give a". We will focus on concrete, specified reductions. That means that the adversary $\mathcal{B}$ not only exists, but is fully specified — minus the details of $\mathcal{A}$ — within the proof. In particular, if you give someone $\mathcal{A}$ then $\mathcal{B}$ becomes runnable. Runnable reductions are generally speaking easier to interpret when it comes to implied security guarantees. They even allow us to use the human-ignorance model [**rogaway2006formalizing**] which, roughly, states that a reduction even to a mathematically easy assumption can still be meaningful. (We will revisit this particular issue with an example in the context of collision resistance.) An example of a non-runnable $\mathcal{B}$ would be one that includes some constant value that we know exists, but don't know an exact value for. This comes up in various arguments, and can cause problems in interpreting the reduction in terms of concrete security. This issue is subtle and we will revisit it.

The takeaway here being that one interprets "we give a" to mean runnable adversaries that are specified fully in the proof. (Or when brevity is at stake, specified to a leave of detail that the average reader could specify it in detail easily.) When we deviate from this convention we should remark on it.

Exhaustive key search. We now ask whether we can lower-bound (T)KR security in general. We do this by providing a *generic* attack, or an attack that works against any cipher. One such generic attack is the exhaustive key search attack. The pseudocode is shown in Figure **??**. At a high level, this attack simply chooses a message at random, queries the Fn oracle to get the corresponding ciphertext, and then brute-force searches through the entire keyspace until it finds the key that outputs the correct ciphertext.

We know that $\mathbf{Adv}^{\mathrm{kr}}_{\mathcal{E}}(\mathcal{A}_{\mathrm{eks}}) = 1$ since a consistent key is guaranteed to exist. However, notice that finding $\mathbf{Adv}^{\mathrm{tkr}}_{\mathcal{E}}(\mathcal{A}_{\mathrm{eks}})$ is trickier: $\mathcal{A}_{\mathrm{eks}}$ might return a consistent key that is not necessarily the target key. For instance, this attack would not work on the identity map cipher since every key is consistent. For "real" ciphers, we expect this to be close to 1. The worst-case running time for this attack is $|\mathcal{K}|$, while the expected running time is $|\mathcal{K}|/2$.

Computational security. Computational security presents a large paradigm shift from previous notions. It focuses on computationally-bound adversaries. For instance, exhaustive key search attack is clearly not computationally efficient and thus considered a weak attack. We measure computational costs by assuming abstract unit costs of (most) operations. This is course-grained but useful for our purposes.

We have shown previously that TKR is not a good security notion, but we now ask whether KR is an improved definition. In particular, the identity cipher has been shown to be secure under TKR security, yet it is insecure under KR security, which is clearly an improvement. However, KR does not imply message confidentiality: it is a necessary but not sufficient goal. We now move on to a very different notion of security.

PRP and PRF security. A standard goal for cipher security is security in the sense of pseudorandom permutations and/or pseudorandom functions. For simplicity, we will focus on **block ciphers**, which for keyspace $\mathcal{K} = \{0,1\}^k$ and message space $\mathcal{M} = \{0,1\}^n$ are defined by $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. Let $Perm(n)$ be the set of all permutations on n bits. Notice that since $|\{0,1\}^n| = 2^n$, then $|Perm(n)| = 2^n!$. Let $Func(n,n)$ be the set of all functions from $\{0,1\}^n \to \{0,1\}^n$. We again note that $|Func(n,n)| = (2^n)^{2^n}$.

We now define a **pseudorandom function** (PRF) as a function that is indistinguishable from a random function (RF). At a high level this means that the input-output behavior of some block cipher E_K "looks like" the input-output behavior of a random function. The games for PRF security are provided in Figure **??**. There are two games defined for PRF security: PRF1 and PRF0. In

```
$\underline{\mathrm{TKR}_{\mathcal{E}}^{\mathcal{A}}}$
$K \leftarrow\!\!\$\ \mathcal{K}$
$K^* \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Fn}}$
Ret $(K = K^*)$

$\underline{\mathrm{Fn}(M)}$
$C \leftarrow E_K(M)$
Ret $C$
```

Figure 3: The target key recovery game.

```
$\underline{\mathrm{KR}_{\mathcal{E}}^{\mathcal{A}}}$
$\mathsf{win} \leftarrow \mathsf{false}$
$K \leftarrow\!\!\$\ \mathcal{K}$
$K^* \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Fn}}$
For $M \in \mathcal{X}$:
    If $E_{K^*}(M) \neq E_K(M)$ then
        $\mathsf{win} \leftarrow \mathsf{false}$
Return $\mathsf{win}$

$\underline{\mathrm{Fn}(M)}$
$\mathsf{win} \leftarrow \mathsf{true}$
$\mathcal{X} \leftarrow \mathcal{X} \cup \{M\}$
$C \leftarrow E_K(M)$
Ret $C$
```

Figure 4: The key recovery game.

```
$\underline{\mathcal{A}_{\mathrm{Fn}}^{\mathrm{eks}}}$
$M \leftarrow\!\!\$\ \mathcal{M}$
$C \leftarrow \mathrm{Fn}(M)$
For $K^* \in \mathcal{K}$ do:
    If $C = E(K^*, M)$ then
        Return $K^*$
Return $\perp$
```

Figure 5: The exhaustive key search attack.

```
$\underline{\mathrm{PRF1}_{\mathcal{E}}^{\mathcal{A}}}$
$K \leftarrow\!\!\$\ \mathcal{K}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Fn}}$
Return $b'$

$\underline{\mathrm{Fn}(M)}$
Return $E_K(M)$
```

```
$\underline{\mathrm{PRF0}_{\mathcal{E}}^{\mathcal{A}}}$
$\rho \leftarrow\!\!\$\ \mathrm{Func}(n, n)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Fn}}$
Return $b'$

$\underline{\mathrm{Fn}(M)}$
Return $\rho(M)$
```

Figure 6: The PRF security games.

PRF1, the adversary has access to the Fn oracle that returns the output from the block cipher E_K. However, in PRF0 the adversary instead receives the output from a random function ρ when it queries the Fn oracle. The adversary $\mathcal{A}$ does not know in which game it is playing and must query the Fn oracle to distinguish between E_K and ρ. $\mathcal{A}$ returns a bit signifying which game it believes it is in. The PRF advantage for $\mathcal{A}$ is defined as

$$\mathbf{Adv}_{\mathcal{E}}^{\mathrm{prf}}(\mathcal{A}) = \left|\Pr\left[\,\mathrm{PRF1}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{PRF0}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow 1\,\right]\right|.$$

The adversary $\mathcal{A}$ wins if the probability that $\mathcal{A}$ outputs 1 in game $\mathrm{PRF1}_{\mathcal{E}}^{\mathcal{A}}$ is far greater than the probability that it outputs 1 in game $\mathrm{PRF0}_{\mathcal{E}}^{\mathcal{A}}$. In particular, notice that if $\mathcal{A}$ simply always output 1, $\mathbf{Adv}_{\mathcal{E}}^{\mathrm{prf}}(\mathcal{A})$ would be 0 as expected, since $\mathcal{A}$ did not successfully distinguish $\mathcal{E}$ from a random function.

Just as we provided a generic attack for TKR security using the exhaustive key search attack, is there a generic distinguishing attack for any cipher? One interesting observation is that for a given key, a block cipher E_K is a permutation, meaning that two different inputs could not produce the same output. (If this were not the case, decryption would be impossible.) However, a random function simply chooses outputs at random, so it is entirely possible for two different inputs to produce the same output. The probability of choosing q values at random from $\{0,1\}^n$ and for two of these values to be the same is approximately $\frac{q^2}{2^n}$. This is colloquially known as the **birthday bound**, since it implies that the number of people expected to produce two individuals with the same birthday is far fewer than what one would expect.

If $\mathcal{A}$ were to query its Fn oracle enough times, eventually the probability that a repeat value is produced would be large enough and $\mathcal{A}$ could then check to see if such a repeat value exists. If it does, then clearly $\mathcal{A}$ must be in game $\mathrm{PRF0}_{\mathcal{E}}^{\mathcal{A}}$; otherwise, $\mathcal{A}$ can assume it is in game $\mathrm{PRF1}_{\mathcal{E}}^{\mathcal{A}}$. This attack is called the **birthday attack**. The pseudocode for this adversary is defined below.

```
$\underline{\textbf{adversary } A_{\mathrm{bday}}^{\mathrm{Fn}}}$
Let $M_1, M_2, \cdots, M_q \leftarrow \{0,1\}^n$ be distinct
For $i = 1, \cdots, q$ do $C_i \leftarrow \mathrm{Fn}(M_i)$
If $\exists i \neq j$ such that $C_i = C_j$ then return 0
Else return 1
```

Notice that in game $\mathrm{PRF1}_{\mathcal{E}}^{\mathcal{A}}$ all output values will be distinct, so A_{bday} will always return 1 and thus $\Pr\left[\,\mathrm{PRF1}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow 1\,\right] = 1$. The probability that A_{bday} returns 1 in game $\mathrm{PRF0}_{\mathcal{E}}^{\mathcal{A}}$ is the probability that all values chosen at random from the random function will be distinct, which is $1 - \frac{q^2}{2^n}$. The PRF-advantage of $A_{\mathrm{bday}}^{\mathrm{Fn}}$ is then defined as

$$\mathbf{Adv}_{\mathcal{E}}^{\mathrm{prf}}(\mathcal{A}) \geq 1 - \left(1 - \frac{q^2}{2^n}\right) = \frac{q^2}{2^n}.$$

```
$\mathrm{PRP1}_{\mathcal{E}}^{\mathcal{A}}$
$K \leftarrow\!\$\ \mathcal{K}$
$b' \leftarrow\!\$\ \mathcal{A}^{\mathrm{Fn}}$
Return $b'$

$\mathrm{Fn}(M)$
Return $E_K(M)$
```

```
$\mathrm{PRP0}_{\mathcal{E}}^{\mathcal{A}}$
$\pi \leftarrow\!\$\ \mathrm{Perm}(n)$
$b' \leftarrow\!\$\ \mathcal{A}^{\mathrm{Fn}}$
Return $b'$

$\mathrm{Fn}(M)$
Return $\pi(M)$
```

Figure 7: The PRP security games.

We define a **pseudorandom permutation** (PRP) as a function that is indistinguishable from a random permutation (RP). The games for PRP security are provided in Figure **??**. These games work similarly to the PRF games but now utilize a random permutation in $\mathrm{PRP0}_{\mathcal{E}}^{\mathcal{A}}$ rather than a random function. The PRP advantage for $\mathcal{A}$ is defined as

$$\mathbf{Adv}_{\mathcal{E}}^{\mathrm{prp}}(\mathcal{A}) = \left|\Pr\left[\,\mathrm{PRP1}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{PRP0}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow 1\,\right]\right| .$$

Considering that these notions are similar, can we relate them to each other? Intuitively, there is no difference between a random function and a random permutation when observing only a few input-output pairs. We formalize this intuition with the following lemma.

Lemma 1 (PRF-PRP Switching Lemma) *Let $\mathcal{E}$ be a cipher with ciphertext space $\{0,1\}^n$. Let $\mathcal{A}$ be an adversary making at most q queries. Then*

$$\left|\Pr\left[\,\mathrm{PRF0}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{PRP0}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow 1\,\right]\right| \le \frac{q^2}{2^n} .$$

The intuition for the following proof is that if you have oracle access to a random function or a random permutation, then you need to make enough queries to witness a collision, as determined by the birthday bound.

One's first instinct might be to bound the difference using a conditioning argument. For instance, let Dist be the event that in game $\mathrm{PRF0}_{\mathcal{E}}^{\mathcal{A}}$ all values returned from oracle Fn are distinct. Then one might say that $\Pr\left[\,\mathrm{PRP1}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow 1\,\right] = \Pr\left[\,\mathrm{PRP0}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow 1|\mathsf{Dist}\,\right]$. However, this is incorrect and in fact $\Pr\left[\,\mathrm{PRP1}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow 1\,\right] \neq \Pr\left[\,\mathrm{PRP0}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow 1|\mathsf{Dist}\,\right]$. Refer to [**bellare2006multi**] for further technical details.

To correctly prove this, we will instead use a game-playing argument. We first provide the following definition and lemma.

Definition 2 *Games* G1 *and* G2 *are **identical-until-bad** if they both contain a* bad *flag and their code differs only in statements following the setting of* bad *to* true.

Lemma 2 (Fundamental Lemma of game playing [bellare2006security]) *Let* G, H *be games that are identical-until-bad and y be any value. Then*

$$\left|\,\Pr\left[\,\mathrm{G} \Rightarrow y\,\right] - \Pr\left[\,\mathrm{H} \Rightarrow y\,\right]\right| \le \Pr\left[\,\mathrm{H} \text{ sets } \mathsf{bad}\,\right] = \Pr\left[\,\mathrm{G} \text{ sets } \mathsf{bad}\,\right] .$$

Proof of Lemma ??: We define the games in Figure **??**. Notice that the output from game G0 has an identical distribution to that of game $\mathrm{PRP0}_{\mathcal{E}}^{\mathcal{A}}$. The only difference between them is that game $\mathrm{PRP0}_{\mathcal{E}}^{\mathcal{A}}$ chooses a random permutation and returns the output from that, while game G0 chooses unique values at random as $\mathcal{A}$ makes queries to Fn. Then $\Pr\left[\,\mathrm{PRP0}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow 1\,\right] = \Pr\left[\,\mathrm{G0}\,\right]$. Game G1 includes the boxed statement and also has an identical output distribution to that of game

```
G0
$b' \leftarrow\$ \mathcal{A}^{\mathrm{Fn}}$
Return $b'$

Fn($M$)
If $\mathtt{F}[M] = \perp$ then
    $\mathtt{F}[M] \leftarrow\$ \{0,1\}^n \setminus \mathtt{F}$
Return $\mathtt{F}[M]$
```

```
$\boxed{\text{G1}}$    G2
$b' \leftarrow\$ \mathcal{A}^{\mathrm{Fn}}$
Return $b'$

Fn($M$)
$C \leftarrow\$ \{0,1\}^n$
If $C \in \mathtt{F}$ then
    bad ← true
    $\boxed{C \leftarrow\$ \{0,1\}^n \setminus \mathtt{F}}$
$\mathtt{F}[M] \leftarrow C$
Return $\mathtt{F}[M]$
```

Figure 8: The games for the switching lemma (Lemma **??**).

G0. It simply chooses an output value at random, and if it detects a repeat it then chooses another unique value. In the case of a repeat, it also sets the bad flag to true. We then have that $\Pr[\,\text{G1}\,] = \Pr[\,\text{G0}\,]$. We next transition to game G2 and notice that G1 and G2 are **identical-until-bad**. The Fundamental Lemma of game playing then states that $\Pr[\,\text{G1}\,] \leq \Pr[\,\text{G2}\,] + \Pr[\,\textsf{bad} \text{ set to } \textsf{true}\,]$. Game G2 returns values chosen at random and allows for repeat values, so it has an identical output distribution to $\text{PRF0}^{\mathcal{A}}_{\mathcal{E}}$. This means $\Pr\left[\,\text{PRF0}^{\mathcal{A}}_{\mathcal{E}} \Rightarrow 1\,\right] = \Pr[\,\text{G2}\,]$. Finally, the probability that bad is set to true is the probability that a random value is chosen by Fn such that it is not distinct, which is bounded by the birthday bound. We then have the following:

$$\begin{aligned}
&\left|\Pr\left[\,\text{PRP0}^{\mathcal{A}}_{\mathcal{E}} \Rightarrow 1\,\right] - \Pr\left[\,\text{PRF0}^{\mathcal{A}}_{\mathcal{E}} \Rightarrow 1\,\right]\right| \\
&\quad = \left|\Pr[\,\text{G0}\,] - \Pr\left[\,\text{PRF0}^{\mathcal{A}}_{\mathcal{E}} \Rightarrow 1\,\right]\right| \\
&\quad = \left|\Pr[\,\text{G1}\,] - \Pr\left[\,\text{PRF0}^{\mathcal{A}}_{\mathcal{E}} \Rightarrow 1\,\right]\right| \\
&\quad \leq \left|\Pr[\,\text{G2}\,] + \Pr[\,\textsf{bad} \text{ set to } \textsf{true}\,] - \Pr\left[\,\text{PRF0}^{\mathcal{A}}_{\mathcal{E}} \Rightarrow 1\,\right]\right| \\
&\quad = \Pr[\,\textsf{bad} \text{ set to } \textsf{true}\,] \\
&\quad \leq \frac{q^2}{2^n}
\end{aligned}$$

▌

3.1 Building PRPs from PRFs

We've seen how it is hard to distinguish between a PRF and a PRP. In the previous section, we showed that it is hard to distinguish between a PRF and a PRP. However, as in the use case of block ciphers for length-preserving encryption and decryption, we specifically require a PRP for correctness purposes (in fact, we want an invertible PRP). This is to ensure that a single ciphertext does not have ambiguous decryptions. A natural question to ask is does the existence of a PRF imply the existence of a PRP, or more constructively, can we build a PRP given a PRF?

In this section, we will show that it is possible to build a PRP from a PRF. We will examine one such construction called a Feistel network which is used in the construction of many early block ciphers, including DES (Data Encryption Standard), the first standardized block cipher.

Feistel networks A Feistel round transforms an arbitrary function into a permutation. Define

$\mathcal{E} : \{0,1\}^{2n} \to \{0,1\}^{2n}$ as the permuation constructed using one Feistel round with function $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. The function F is known as the round function of the Feistel network. This construction is depicted in Figure **??** where $2n$ length inputs and outputs are split into left and right parts of length n notated L and R, respectively. Function F is notated with fixed key K as $F_K : \{0,1\}^n \to \{0,1\}^n$. The Feistel round is then

$$\begin{aligned} L_1 &\leftarrow R_0 \\ R_1 &\leftarrow L_0 \oplus F_K(R_0)\,. \end{aligned}$$

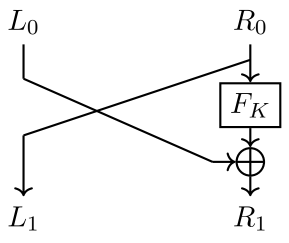

Figure 9: A single round of a Feistel network.

Example 3 [1-round Feistel network is a permutation] First, let us see why the Feistel construction builds a permutation. Notice that as long as the round function F_K is defined on all inputs from domain $\{0,1\}^n$ then $\mathcal{E}$ is defined on all inputs from domain $\{0,1\}^{2n}$. Next, we show that any output of $\mathcal{E}$, $L_1 \,\|\, R_1$, corresponds to a unique input $L_0 \,\|\, R_0$. Even though F_K is not assumed to be invertible, since the input to F_K, R_0, is passed directly to the output as L_1, we can invert as follows

$$\begin{aligned} L_0 &\leftarrow R_1 \oplus F_K(L_1) \\ R_0 &\leftarrow L_1\,. \end{aligned}$$

These two properties together mean $\mathcal{E}$ is a complete permutation on $\{0,1\}^{2n}$.

Example 4 [1-round Feistel network is not a PRP] Next, we show that a 1-round Feistel network $\mathcal{E}$ is not a PRP. Consider the following PRP adversary $\mathcal{A}$ for domain $\{0,1\}^{2n}$:

```
adversary $\mathcal{A}^{\mathrm{Fn}}$
$(L_1, R_1) \leftarrow \mathrm{Fn}(0^{2n})$
Return $L_1 = 0^n$
```

In PRP1, where the oracle Fn returns $\mathcal{E}_K(0^{2n})$, $\mathcal{A}$ returns 1 with probability 1. In PRP0, where oracle Fn returns a random value, the probability $\mathcal{A}$ returns 1 is the probability that the first n bits of the random output are 0. Thus,

$$\begin{aligned} \mathbf{Adv}^{\mathrm{prp}}_{\mathcal{E}}(\mathcal{A}) &= \left|\Pr\left[\,\mathrm{PRP1}^{\mathcal{A}}_{\mathcal{E}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{PRP0}^{\mathcal{A}}_{\mathcal{E}} \Rightarrow 1\,\right]\right| \\ &\geq 1 - \frac{1}{2^n}\,. \end{aligned}$$

We see that it is trivial to distinguish a 1-round Feistel network from a random permutation, since the second half of the input is mapped directly to the first half of the output. Does the randomness of the permutation improve by performing numerous Feistel rounds? One way to construct a multi-round Feistel network by feeding the outputs of the previous round as inputs to the next round, and where each round is keyed with a different unique key. To use only a single key, we consider an alternate construction, where the round function F in each Feistel round has a different domain, $F : \{0,1\}^k \times \{0,1\}^{2n} \to \{0,1\}^n$. As before, the round function will take in the second half of the input, but it will also take in an n-bit round counter. To stack Feistel rounds into a multi-round Feistel network, we simply ensure that each round uses a different round counter. Figure **??** depicts a 3-round Feistel network constructed in this manner.

Example 5 A 2-round Feistel network is not a PRP. ≪**Scribe 3.1:** Add as homework problem≫

3-round Feistel network is a PRP Next, we show that given PRF security of the underlying round function, a 3-round Feistel network is a PRP. Intuitively, the proof will follow the intuition that 3 rounds of Feistel allows the round function to properly add randomness to both $L_3 = R_2$ and R_3. In particular, we want to show that the output of the $F_K(\langle 2\rangle \parallel R_1)$ and $F_K(\langle 3\rangle \parallel R_2)$ cannot be effectively controlled by adversary $\mathcal{A}$, i.e., $\mathcal{A}$ cannot create collisions on R_1 and R_2.

Theorem 3 *[Luby-Rackoff] Let $\mathcal{E}$ be the 3-round Feistel cipher using round function $F\colon \{0,1\}^k \times \{0,1\}^{2n} \to \{0,1\}^n$. For any $\mathrm{PRP}_{\mathcal{E}}$-adversary $\mathcal{A}$ making at most q queries we give an PRF_F-adversary $\mathcal{B}$ making at most $3q$ queries such that*

$$\mathbf{Adv}^{\mathrm{prp}}_{\mathcal{E}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{B}) + \frac{2q^2}{2^n} + \frac{q^2}{2^{2n}}\ .$$

≪**Scribe 3.2:** How are we doing cites? (Luby-Rackoff)≫

Proof: Without loss of generality, we consider $\mathcal{A}$ that make only unique queries to oracle Fn. ≪**Scribe 3.3:** Be more formal about why this okay?≫ We bound the advantage of PRP adversary $\mathcal{A}$ by bounding the advantage of each of a series of game hops. Pseudocode for the games is given in Figure **??**.

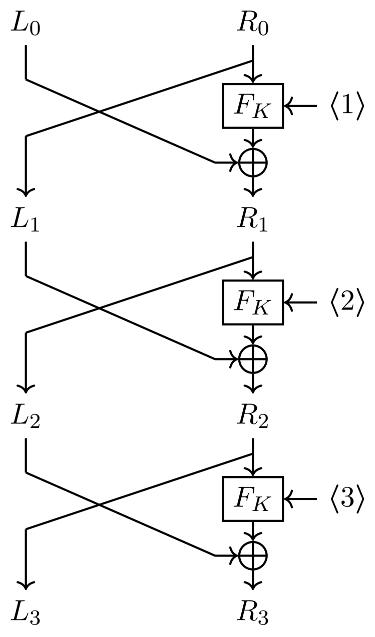

Figure 10: A 3-round Feistel network.

Game G0 is constructed exactly as $\mathrm{PRP1}^{\mathcal{A}}_{\mathcal{E}}$; the oracle pseudocode expands out the computation of the 3-round Feistel network $\mathcal{E}$:

$$\Pr\left[\,\mathrm{PRP1}^{\mathcal{A}}_{\mathcal{E}} \Rightarrow 1\,\right] = \Pr\left[\,\mathrm{G0} \Rightarrow 1\,\right].$$

Game G1 replaces the round function F with a random function ρ mapping from $\{0,1\}^{2n} \to \{0,1\}^n$. ≪**Scribe 3.4:** Should PRF security game be defined with respect to different domain and range space, m and n?≫ We bound the ability to distinguish between G0 and G1 by the PRF security of F. Consider the following PRF adversary $\mathcal{B}$ which runs $\mathcal{A}$ with a simulated oracle FnSim.

```
$\mathcal{B}^{\mathrm{Fn}}$
$K \leftarrow\!\$\ \{0,1\}^k$
$b' \leftarrow\!\$\ \mathcal{A}^{\mathrm{FnSim}}$
Return $b'$

$\mathrm{FnSim}(M)$
$L_1 \leftarrow R_0$
$R_1 \leftarrow L_0 \oplus \mathrm{Fn}(\langle 1\rangle \parallel R_0)$
$L_2 \leftarrow R_1$
$R_2 \leftarrow L_1 \oplus \mathrm{Fn}(\langle 2\rangle \parallel R_1)$
$L_3 \leftarrow R_2$
$R_3 \leftarrow L_2 \oplus \mathrm{Fn}(\langle 3\rangle \parallel R_2)$
Return $L_3 \parallel R_3$
```

Adversary $\mathcal{B}$ simulates FnSim by running the 3-round Feistel network but replacing the round function with a call to

```
G0
$K \leftarrow\$ \{0,1\}^k$
$b' \leftarrow\$ \mathcal{A}^{\mathrm{Fn}}$
Return $b'$

Fn($M$)
$(L_0, R_0) \leftarrow M$
$L_1 \leftarrow R_0$
$R_1 \leftarrow L_0 \oplus F_K(\langle 1 \rangle \parallel R_0)$
$L_2 \leftarrow R_1$
$R_2 \leftarrow L_1 \oplus F_K(\langle 2 \rangle \parallel R_1)$
$L_3 \leftarrow R_2$
$R_3 \leftarrow L_2 \oplus F_K(\langle 3 \rangle \parallel R_2)$
Return $L_3 \parallel R_3$
```

```
G1
$\rho \leftarrow\$ \mathrm{Func}(2n, n)$
$b' \leftarrow\$ \mathcal{A}^{\mathrm{Fn}}$
Return $b'$

Fn($M$)
$(L_0, R_0) \leftarrow M$
$L_1 \leftarrow R_0$
$R_1 \leftarrow L_0 \oplus \rho(\langle 1 \rangle \parallel R_0)$
$L_2 \leftarrow R_1$
$R_2 \leftarrow L_1 \oplus \rho(\langle 2 \rangle \parallel R_1)$
$L_3 \leftarrow R_2$
$R_3 \leftarrow L_2 \oplus \rho(\langle 3 \rangle \parallel R_2)$
Return $L_3 \parallel R_3$
```

```
$\boxed{\text{G2}}$ G3
$b' \leftarrow\$ \mathcal{A}^{\mathrm{Fn}}$
Return $b'$

Fn($M$)
$(L_0, R_0) \leftarrow M$
$L_1 \leftarrow R_0$
If $\mathtt{F}[\langle 1 \rangle \parallel R_0] = \bot$ then
    $\mathtt{F}[\langle 1 \rangle \parallel R_0] \leftarrow\$ \{0,1\}^n$
$X_1 \leftarrow \mathtt{F}[\langle 1 \rangle \parallel R_1]$
$R_1 \leftarrow L_0 \oplus X_1$
$L_2 \leftarrow R_1$
$X_2 \leftarrow\$ \{0,1\}^n$
If $\mathtt{F}[\langle 2 \rangle \parallel R_1] \neq \bot$ then
    bad $\leftarrow$ true
    $\boxed{X_2 \leftarrow \mathtt{F}[\langle 2 \rangle \parallel R_1]}$
$\mathtt{F}[\langle 2 \rangle \parallel R_1] \leftarrow X_2$
$R_2 \leftarrow L_1 \oplus X_2$
$L_3 \leftarrow R_2$
$X_3 \leftarrow\$ \{0,1\}^n$
If $\mathtt{F}[\langle 3 \rangle \parallel R_2] \neq \bot$ then
    bad $\leftarrow$ true
    $\boxed{X_3 \leftarrow \mathtt{F}[\langle 3 \rangle \parallel R_2]}$
$\mathtt{F}[\langle 3 \rangle \parallel R_2] \leftarrow X_3$
$R_3 \leftarrow L_2 \oplus X_3$
Return $L_3 \parallel R_3$
```

```
G3 $\boxed{\text{G4}}$
$b' \leftarrow\$ \mathcal{A}^{\mathrm{Fn}}$
Return $b'$

Fn($M$)
$(L_0, R_0) \leftarrow M$
$L_1 \leftarrow R_0$
If $\mathtt{F}[\langle 1 \rangle \parallel R_0] = \bot$ then
    $\mathtt{F}[\langle 1 \rangle \parallel R_0] \leftarrow\$ \{0,1\}^n$
$X_1 \leftarrow \mathtt{F}[\langle 1 \rangle \parallel R_1]$
$R_1 \leftarrow L_0 \oplus X_1$
$L_2 \leftarrow R_1$
$X_2 \leftarrow\$ \{0,1\}^n$
If $\mathtt{F}[\langle 2 \rangle \parallel R_1] \neq \bot$ then
    bad $\leftarrow$ true
$\mathtt{F}[\langle 2 \rangle \parallel R_1] \leftarrow X_2$
$R_2 \leftarrow L_1 \oplus X_2$; $\boxed{R_2 \leftarrow\$ \{0,1\}^n}$
$L_3 \leftarrow R_2$
$X_3 \leftarrow\$ \{0,1\}^n$
If $\mathtt{F}[\langle 3 \rangle \parallel R_2] \neq \bot$ then
    bad $\leftarrow$ true
$\mathtt{F}[\langle 3 \rangle \parallel R_2] \leftarrow X_3$
$R_3 \leftarrow L_2 \oplus X_3$; $\boxed{R_3 \leftarrow\$ \{0,1\}^n}$
Return $L_3 \parallel R_3$
```

Figure 11: Games for proof of 3-round Feistel network as PRP (Theorem **??**)

its own oracle Fn. In PRF0, where $\mathcal{B}$'s oracle Fn acts as the round function F, $\mathcal{B}$ runs exactly G0. In PRF1, where Fn acts as a random function ρ, $\mathcal{B}$ runs exactly G1. Thus, we have

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{B}) &= \left|\Pr\left[\,\mathrm{PRF1}^{\mathcal{B}}_{F} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{PRF0}^{\mathcal{B}}_{F} \Rightarrow 1\,\right]\right| \\ &= \left|\Pr\left[\,\mathrm{G1} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{G0} \Rightarrow 1\,\right]\right|.\end{aligned}$$

In games G2 and G3, we use a similar trick to as in our proof of the PRF-PRP switching lemma. Game G2 replaces the random function ρ with a lazy evaluation of a random function using a table `F`. When the random function is queried on an input, the value in `F` keyed by the input is returned. If there is no such value, meaning the input has not been previously queried, a new random value is generated, returned, and stored in `F` for future queries. Thus, lazily building out `F` is equivalent to using random function ρ:

$$\Pr\left[\,\mathrm{G1} \Rightarrow 1\,\right] = \Pr\left[\,\mathrm{G2} \Rightarrow 1\,\right].$$

Game G3 generates fresh random values on every input to the second and third round functions. This is in contrast to G2 where a fresh random value is only returned for inputs that have never been seen. Thus, the difference between G2 and G3 occur when repeat inputs are used with the second or third round functions, which corresponds to repeat values of R_1 and R_2, respectively. Intuitively, since we are only considering adversaries $\mathcal{A}$ that make unique queries to Fn, finding repeat values of R_1 and R_2 is hard because they both include randomness from the random function. The pseudocode in Figure **??** captures the event of a repeat query by a setting a bad flag. The only difference between G2 and G3 occur after the bad flag is set; G2 returns the consistent value from the look-up table `F`, while G3 returns a fresh random value. Then by the fundamental lemma of game-playing,

$$|\Pr[\,\mathrm{G3} \Rightarrow 1\,] - \Pr[\,\mathrm{G2} \Rightarrow 1\,]| \leq \Pr[\,\mathrm{G3 \text{ sets } \mathsf{bad}}\,]\,.$$

Game G4 is the same as G3 except R_2 and R_3 are set to fresh random values. In G3, we have $R_2 \leftarrow L_1 \oplus X_2$ and $R_3 \leftarrow L_2 \oplus X_3$ for fresh random values X_2, X_3. Thus, the probability space of R_2 and R_3 are the same and G4 is identical to G3:

$$\Pr[\,\mathrm{G3} \Rightarrow 1\,] = \Pr[\,\mathrm{G4} \Rightarrow 1\,]\,.$$

Importantly, this also implies

$$\Pr[\,\mathrm{G3 \text{ sets } \mathsf{bad}}\,] = \Pr[\,\mathrm{G4 \text{ sets } \mathsf{bad}}\,]\,.$$

Notice that G4 simply returns a fresh random string of length $2n$ on every query to oracle Fn. Since we assume $\mathcal{A}$ only makes unique queries to Fn, G4 is simulating a perfect random function, and thus is equivalent to the PRF0 game:

$$\Pr[\,\mathrm{G4} \Rightarrow 1\,] = \Pr\left[\,\mathrm{PRF0}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow 1\,\right].$$

And by the PRF-PRP switching lemma,

$$\left|\Pr\left[\,\mathrm{PRF0}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{PRP0}_{\mathcal{E}}^{\mathcal{A}} \Rightarrow 1\,\right]\right| \leq \frac{q^2}{2^{2n}}\,.$$

Lastly, we can bound the probability G4 sets bad. First consider the probability G4 sets bad on query i; call this event W_i. This event can be considered as two separate events depending on where the bad flag is set: denote the event that R_1 collides as W_i^1 and the event that R_2 collides as W_i^2.

Let's first bound the easier case, the probability of W_i^2. The probability of $R_2 = L_1 \oplus X_2$ colliding with a previous R_2 on query i is bounded by $q/2^n$ since X_2 is a fresh random string.

Bounding the probability of W_i^1 is more nuanced. At first glance it appears that the logic should follow symmetrically to above where $R_1 = L_0 \oplus X_1$ for random X_1. To bound the probability of collision to $q/2^n$, we must argue that X_1 is independent of the inputs to Fn. In the previous case, X_2 is freshly sampled on every query to Fn so independence is trivially satisfied. This is not the case for X_1. However, we can argue that $\mathcal{A}$'s queries are dependent only on $\mathcal{A}$'s random coins and the previous outputs of Fn. Since in G4, the outputs of Fn are independently drawn random samples, we have that the outputs of Fn are independent of X_1. ≪**Scribe 3.5:** May need to argue this more formally?≫

Thus, by two applications of the union bound, we have

$$\begin{aligned}
\Pr[\,\mathrm{G4 \text{ sets } \mathsf{bad}}\,] &\leq \sum_i^q \Pr[\,W_i\,] \\
&\leq \sum_i^q \Pr\left[\,W_i^1\,\right] + \Pr\left[\,W_i^2\,\right] \\
&\leq \sum_i^q \frac{q}{2^n} + \frac{q}{2^n} \\
&= \frac{2q^2}{2^n}.
\end{aligned}$$

Finally, to put it all together,

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{prp}}_{\mathcal{E}}(\mathcal{A}) &= \left|\Pr\left[\,\mathrm{PRP1}^{\mathcal{A}}_{\mathcal{E}}\,\right] - \Pr\left[\,\mathrm{PRP0}^{\mathcal{A}}_{\mathcal{E}}\,\right]\right| \\
&= \left|\Pr\left[\,\mathrm{G0}\,\right] - \Pr\left[\,\mathrm{PRP0}^{\mathcal{A}}_{\mathcal{E}}\,\right]\right| \\
&\le \left|\Pr\left[\,\mathrm{G1}\,\right] + \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{B}) - \Pr\left[\,\mathrm{PRP0}^{\mathcal{A}}_{\mathcal{E}}\,\right]\right| \\
&= \left|\Pr\left[\,\mathrm{G2}\,\right] + \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{B}) - \Pr\left[\,\mathrm{PRP0}^{\mathcal{A}}_{\mathcal{E}}\,\right]\right| \\
&\le \left|\Pr\left[\,\mathrm{G3}\,\right] + \Pr\left[\,\mathrm{G3\ sets\ \mathsf{bad}}\,\right] + \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{B}) - \Pr\left[\,\mathrm{PRP0}^{\mathcal{A}}_{\mathcal{E}}\,\right]\right| \\
&= \left|\Pr\left[\,\mathrm{G4}\,\right] + \Pr\left[\,\mathrm{G4\ sets\ \mathsf{bad}}\,\right] + \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{B}) - \Pr\left[\,\mathrm{PRP0}^{\mathcal{A}}_{\mathcal{E}}\,\right]\right| \\
&\le \left|\Pr\left[\,\mathrm{G4}\,\right] + \frac{2q^2}{2^n} + \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{B}) - \Pr\left[\,\mathrm{PRP0}^{\mathcal{A}}_{\mathcal{E}}\,\right]\right| \\
&\le \left|\Pr\left[\,\mathrm{PRP0}^{\mathcal{A}}_{\mathcal{E}}\,\right] + \frac{q^2}{2^{2n}} + \frac{2q^2}{2^n} + \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{B}) - \Pr\left[\,\mathrm{PRP0}^{\mathcal{A}}_{\mathcal{E}}\,\right]\right| \\
&= \frac{q^2}{2^{2n}} + \frac{2q^2}{2^n} + \mathbf{Adv}^{\mathrm{prf}}_{F}(\mathcal{B})
\end{aligned}$$

≪**Scribe 3.6:** Additional Exercise Ideas≫

- 2-round Feistel is not a PRP
- Show security of 3-round Feistel with 3 different keys
- What happens when same key is used across Feistel rounds? (I don't know)
- Show 3-round Feistel is not Strong PRP (Strong PRP gets access to both forward and inverse oracle)
- Show 4-round Feistel is Strong PRP

Connection to card shuffling algorithms ≪**Scribe 3.7:** TODO≫

4 Block Cipher Design and Cryptanalysis

So far we've seen some theoretical ways to construct block ciphers, namely Feistel networks using round functions that are as secure are PRFs. There are other ways to build block ciphers such as the Even-Mansour [**EvenMansour**] construction from a single known PRP. It is basically of the form:

$$E_{<K_1,K_2>}(M) = F(M \oplus K_1) \oplus K_2$$

Here, K_1, K_2 are the keys used for Message M and F is a PRP that is known (or can be easily obtained) for the Even-Mansour encryption scheme E.

But these kinds of designs can prove to be reductive since the mechanisms to build PRFs in practice is itself unclear. One could try to use actual random functions. But this is untractable for large block sizes, the secret key, in this case, being a random table requiring $n2^n$ bits (For block sizes of n, the lookup table has to have at least 2^n possible n-bit string values to look indistinguishable from a random function for a particular key).

In practice block ciphers are built using a bag of specific design principles that have been developed over the past 60 or so years in response to new cryptoanalytic techniques. It is important to note that block ciphers by themselves are just tools. Like any other tool, they must be used correctly in order for them to satisfy certain security properties that end users might care about.

For example, the identity cipher satisfies all the mathematical properties of a block cipher. But it is of no real use since it doesn't hide the message (in other words, it doesn't provide message confidentiality).

Another example is the one-time pad which we have proved to be perfectly secure. But under closer examination, we see that the one-time pad can easily be broken if the key is reused more than once. (Basically, if one of the messages is known under a known plain text attack, the attacker can retrieve the key)

Here, we will study two kinds of common block cipher constructions DES (Data Encryption Standard) and AES (Advanced Encryption Standard) that are used in practice and use cryptanalysis to evaluate the effectiveness of these ciphers.

4.1 DES: Data Encryption Standard

DES was developed at IBM in the 1970s with the support of the NSA. It has been the single most widely used cipher and was responsible for jump-starting the field of cryptanalysis. The precursor to DES was IBM's block cipher called Lucifer. Certain variants of Lucifer operated on 128-bit blocks using 128-bit keys. The National Bureau of Standards, however, asked for a block cipher that used shorter blocks (64 bits) and shorter keys (56 bits). In response, the IBM team designed a block cipher that met these requirements which eventually became DES. [**BonehShoupBook**].

We will see that reducing the block size creates problems and DES is now considered insecure and should not be used. We will discuss a more secure variant of DES called Triple-DES that has been approved by NIST through to 2030 and is currently in use[**BonehShoupBook**].

r2.5in

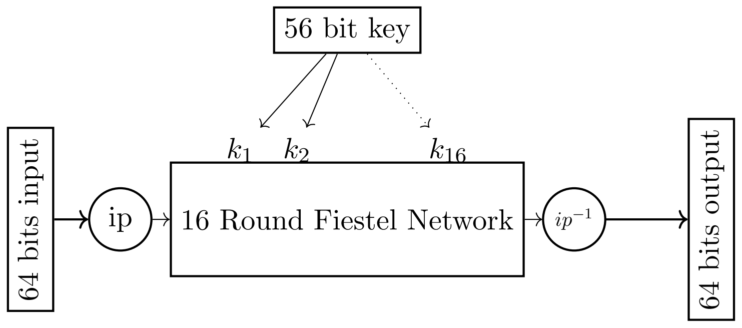

Figure 12: Constructing DES from a fiestel network

Construction of DES

DES uses a Feistel network construction spanning 16 rounds and a different function at each round. DES at a high level takes an input of 64 bits and permutes it in an initial permutation (ip). Afterward, the 64 bits are split into 32-bit parts, L_0, R_0 which are taken as inputs into the first round of the Feistel network. The input key is 56 bits and DES uses a key schedule that expands the 56-bit key into 16, 48-bit round keys which are used in each round of DES. After all the rounds of the Feistel network, DES runs one final permutation that is the inverse of the initial permutation ip^{-1} before returning the output ciphertext.

Let ip be the permutation function and $F : \{0,1\}^{64} \to \{0,1\}^{64}$ the fiestel network function. Then for any message m where ($|m| = 64$ bits) and key k where ($|k| = 56$ bits), DES can be defined as follows:

$$E_{DES}(m, k) = ip^{-1}(F(ip(m), k)) = \mathcal{E}$$

For the given fiestel network function F, Let $f_1, f_2, ...f_{16} : \{0,1\}^{32} \to \{0,1\}^{32}$ be the specific round functions of each round of the fiestel permutation and $k_1, k_2, ...k_16$ be the round keys used in each round. Then the round functions in DES can be represented as follows:
$I = L_0||R_0$ and $|L_0| = |R_0|$, in other words L_0 is the first 32 bits of the initial input I and R_0 is the remaining 32 bits

$$\forall_{i=1}^{16} i : L_i \leftarrow R_{i-1}$$
$$R_i \leftarrow L_{i-1} \oplus f_i(R_{i-1}, k_i)$$

DES round functions

Although each round of the fiestel permutation in DES uses a different round function, they follow a similar structure in using the set of auxiliary functions given below:

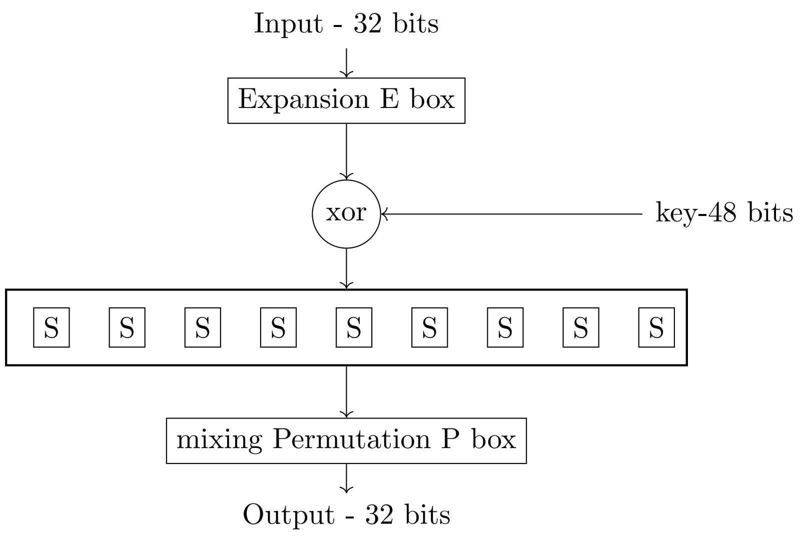

Figure 13: Sbox abstraction

- **Expansion function**(E): The expansion function mainly takes the initial 32-bit input and transforms it into 48 bits by a mixture of permutation and replication of various bits.
- **Mixing Permutation** (P): The mixing permutation maps a 32-bit input to a 32-bit output by mainly rearranging the bits of the input
- **S boxes** ($S_1, S_2 ... S_8$): S boxes are the heart of the round functions and DES uses 8 of them in each round. They act as look up tables that map 6-bit inputs to 4-bit outputs. The DES standard lists these 8 lookup tables, where each table contains 64 entries.

Given these functions, for the key k and input x, the DES round function $f(k, x)$ works as follows:

```
$f(k,x) : \{0,1\}^{48} X \{0,1\}^{32} \rightarrow \{0,1\}^{32}$
$f(k,x) :$
    $t \leftarrow E(x) \oplus k$ $--$ (transforms 32$-$bit input to 48 bits)
    split t into $t_1, t_2 ... t_8$ such that $t = t_1 || t_2 ... t_8$ and $|t_1| = |t_2| = ... |t_8| = 6$ bits
    $\forall i \in \{1, 2, ...8\} : s_i \leftarrow S_i(t_i)$
    $s \leftarrow s_1 || s_2 || ... || s_8$
    return P(s)
```

≪**Scribe 4.1:** homework question on S boxes - How does the choice of S-boxes affect DES. What happens if S boxes are the same in every round? I guess all the round functions would end up becoming the same in the fiestel - not sure about this as expansion/permutation functions could change≫
≪**Scribe 4.2:** Another homework question - Why did they choose the S-box to transform 6 bits to 4 bits? the choice seems aribitrary here on the split≫

It is important to note that the DES round cipher is made up entirely of XORs and bit permutations. The S-boxes are the only operations that introduce non-linearity into the design.

Linear Cryptanalysis

The purpose of linear cryptanalysis is to be able to approximate a given (non-linear) block cipher using a linear expression. These linear estimates of an unknown cipher can be used to develop

successful attacks against the original nonlinear-cipher. We will later look at ways to construct a linear approximation of des to launch a known plaintext attack to retrieve the key.
For a given plaintext P, ciphertext C and key K, a linear expression takes the form of:

$$P[p_1, p_2...p_a] \oplus C[c_1, c_2, ...c_b] = K[k_1, k_2....k_c] \tag{1}$$

Here $p_1, p_2...p_a$, $c_1, c_2, ...c_b$ and $k_1, k_2....k_c$ denote fixed bit positions (example p_1 denotes the 1st position of the plaintext).

The probability that the equation (**??**) holds true for a randomly chosen plaintext and its corresponding ciphertext should deviate from $\frac{1}{2}$. So the effectiveness of the equation can be captured by $|p - \frac{1}{2}|$ where $p = \Pr[\ P[p_1, p_2...p_a] \oplus C[c_1, c_2, ...c_b] = K[k_1, k_2....k_c]\ ]$.

Given an effective linear approximation, it is possible to determine one bit of information about the key using the following algorithm. It's basically a maximum-likelihood estimate.

Given below is the algorithm to estimate this effectiveness/bias.

```
Step 1: Let T be the number of plaintexts such that left side of equation (1) = 0
Step 2: If T > N/2 (N = number of plain texts) -
            then if p > 1/2, guess K[k1, k2 ... kc] = 0
            else guess K[k1, k2 ... kc] = 1
        else
            if p > 1/2, guess K[k1, k2 ... kc] = 1
            else guess K[k1, k2 ... kc] = 0
```

The rate of success for the given algorithm increases with the increase in the number of plaintexts N used or with the increase in effectiveness $|p - \frac{1}{2}|$ for a given linear approximation.

Given below is another way of representing the effectiveness of the linear expression in terms of the number of plaintexts sampled q (The above notation is used in Matsui's original paper [**MatsuiDES**]. For $\forall S \in \{S_k, S_m, S_c\} : S \subsetneq \{1, 2, ...n\}, n =$ size of blockcipher and $X[S] = \oplus_{\ i \in S} X_i$ where X is a bit string.

$$\Pr[\ K[S_k] = \mathrm{Maj}\,(\{M_i[S_m] \oplus C_i[S_c]\}_{i=1}^{q})\ ]$$

Let ϵ be the deviation/bias of this estimate from $\frac{1}{2}$ So

$$p = \frac{1}{2} + \epsilon$$

$$\Pr[\ K[S_k] = \mathrm{Maj}\,(\{M_i[S_m] \oplus C_i[S_c]\}_{i=1}^{q})\ ] = \frac{1}{2} + \epsilon$$

Theorem 4 *Let $\mathcal{E}$ be a cipher such that (**??**) holds with $\epsilon > 0$, and let $K \in \mathcal{K}$. Let $M_1, \ldots, M_q$ be sampled uniformly from $\{0,1\}^n$ and let $C_i = E_K(M_i)$ for $i \in \{1, \ldots, q\}$. Then*

$$\Pr[\ K[S_k] = \mathrm{Maj}\,(\{M_i[S_m] \oplus C_i[S_c]\}_{i=1}^{q})\ ] \geq 1 - e^{-q\epsilon^2/2}\ .$$

Proof: Let's assume $K[S_k] = 0$, this means for $X = (\{M_i[S_m] \oplus C_i[S_c]\}_{i=1}^q)$, so a majority of sampled values resulted in a 0.

$$E[X] = q * (\frac{1}{2} + \epsilon)$$
$$U = \text{Maj}\,(\{M_i[S_m] \oplus C_i[S_c]\}_{i=1}^q)$$

Theorem 5 *[Chernoff bounds] Let $X = \sum_{i=1}^n X_i$, all X_i independent and where $X_i = 1$ with probability p_i and $X_i = 0$ with probability $1 - p_i$. Let $\mu = \boldsymbol{E}[X]$. Then*

$$\Pr[\, X \geq (1+\delta)\mu \,] \leq e^{-\frac{\delta^2}{2+\delta}\mu} \tag{2}$$
$$\Pr[\, X \leq (1+\delta)\mu \,] \leq e^{-\frac{\delta^2}{2}\mu} \tag{3}$$

$$p = \Pr[\, U = 0 \,] = \Pr[\, \text{Maj}\,(\{M_i[S_m] \oplus C_i[S_c]\}_{i=1}^q) \,]$$

The probability of a the sampled values being 1 instead of 0 is 1-p, from chernoff bounds for $\epsilon \geq 0$ we get

$$1 - p = \Pr\left[\, X \geq \frac{q}{2} \,\right] \leq \Pr\left[\, X \geq (1+\epsilon)(\frac{1}{2} + \epsilon)q \,\right] \leq e^{\frac{-\epsilon^2}{(2+\epsilon)}(\frac{1}{2}+\epsilon)q}$$

By reducing this we can get

$$p \geq 1 - 1 - e^{-q\epsilon^2/2}$$

the optimal number of plaintexts needed would be $\frac{1}{\epsilon^2}$ (Then the exponent term becomes a small fraction). From the equation we can also see that p is maximized by either having a large q (using a large sample space) or having a large bias ϵ.

Linear Cryptanalysis on DES

Since the S-boxes are the only non-linear components in DES. Finding an efficient linear approximation of DES hinges on finding a way to express the S-box linearly.

For a given S-box $S_a(a = 1, 2...8)$,

$$\Pr[\, [X[S_x] \oplus Sbox(X)[S_y] = 0] \,] = \frac{1}{2} + \epsilon$$

Since there are 2^6 possible inputs and 2^4 possible outputs to the Sbox, we'll define

$$NS_a(\alpha, \beta) = \#\{x | 0 \leq x \leq 64, (\oplus_{s=0}^5 x[s] \wedge \alpha[s]) = (\oplus_{t=0}^3 S_a[x][t] \wedge \beta[t])\}$$

$NS_a(\alpha, \beta)$ essentially tries to capture the correlation of the input bits and output bits in a linear form (in the form of xors) by exploiting the probability bias based on this linear approximation. This allows estimating bits from a round function. For example $NS_5(16, 15) = 12$ implies that the 4th bit of S_5 coincides with an XORed value of all output bits with probabilityy $\frac{12}{64}$

This linear approximation can be generalized to the entire round function by taking into account the expansion function and the permutation. One key bit can now be recovered using the algorithm described initially.

generalizing linear approximation to all of des Individually assessing the input/output relationship between each of the s-boxes also lets us chain them to obtain the approximation for the entire Feistel network. And the bias on each round can be treated as independent variables. This lets us combine the biases

For linear approximations $X_1, X_2, X_3, ...X_n$, the bias of the combination of this system of linear equations is as follows:

$$\epsilon_{1,2,...n} = 2^{n-1} \prod_{i=1}^{n} \epsilon_i$$

A generalized form of the combined linear estimate is given below:

Lemma 3 *Let X_i for $1 \leq i \leq n$ be indendent random variables with probabilities p_i of being one and $1 - p_i$ of being zero. Then*

$$\Pr\left[\, X_1 \oplus \cdots \oplus X_n = 0 \,\right] = \frac{1}{2} + 2^{n-1} \prod_{i=1}^{n} \left(p_i - \frac{1}{2} \right) .$$

Recovering many bits of des The basic intuition is to realize that we can build partial linear approximations of only a subset of the rounds using the piling up lemma. A combination of these linear approximations should enable us to recover multiple bits of des and brute force the rest. 16 round des breaks with 2^{43} known plaintext/ciphertext pairs.)

Linear cryptanalysis can mainly be used to reduce the search space of keys based on the recovered bits and launch a known plaintext attack.

differential cryptanalysis

Differential cryptanalysis is similar to its linear counterpart in that it exploits the high probability of certain occurrences of plaintext differences and differences into the output. However, it aims to capture the relationship using non-linear expressions.

$$\delta_c = E_k(M + \delta_m) + E_k(M)$$

The aim here is to find δ_m such that δ_c holds for the given expression with a high probability over the given choice of M.

In DES, Multiple of these differentials for s-boxes can be chained together like in the linear case and this lead to recovering bits of the key.

≪**Scribe 4.3:** potential question idea: difference between DES, 2DES and 3DES in terms of security≫

*aes ≪**Scribe 4.4:** TODO: Brief notes about aes cipher≫ ≪**Scribe 4.5:** TODO: add fixed diagram for aes cipher≫

```
$\underline{\text{MR-UMA}^{\mathcal{A}}_{\mathcal{E},p_m,q}}$
$K \leftarrow\$ \mathcal{K}$
For $i = 1$ to $q$
    $M_i \leftarrow_{p_m} \mathcal{M}$
    $C_i \leftarrow E_K(M_i)$
$\hat{E} \leftarrow\$ \mathcal{A}(C_1, \ldots, C_q)$
Ret $\forall_{i=1}^{q} \left(\hat{E}(M_i) = C_i\right)$
```

```
$\underline{\text{G1}}$
$\rho \leftarrow\$ \text{Func}(\mathcal{M})$
For $i = 1$ to $q$
    $M_i \leftarrow_{p_m} \mathcal{M}$
    $C_i \leftarrow \rho(M_i)$
$\hat{E} \leftarrow\$ \mathcal{A}(C_1, \ldots, C_q)$
Ret $\forall_{i=1}^{q} \left(\hat{E}(M_i) = C_i\right)$
```

5 Deterministic Encryption and Frequency Analysis

```
$\underline{\mathcal{A}^*(C_1, \ldots, C_q)}$
Let $c$ be number of unique ciphertexts in $C_1, \ldots, C_q$
Let $\tilde{C}_1, \ldots, \tilde{C}_c$ be unique ciphertexts
Let $N_{\tilde{C}_i}$ be number of occurences of $\tilde{C}_i$
$\hat{E} \leftarrow \arg\max_f \prod_{i=1}^{c} p_m \left(f^{-1}(\tilde{C}_i)\right)^{N_{\tilde{C}_i}}$
Ret $\hat{E}$
```

$$\mathbf{Adv}^{\text{mr-uma}}_{\mathcal{E},p_m,q}(\mathcal{A}) = \Pr\left[\,\text{MR-UMA}^{\mathcal{A}}_{\mathcal{E},p_m,q} \Rightarrow \mathsf{true}\,\right]$$

Theorem 6 *Let $\mathcal{E}$ be a cipher, p_m a message distribution, and $q > 0$. Let $\mathcal{A}^*$ be the frequency analysis* MR-UMA$_{\mathcal{E},q}$*-adversary and $\mathcal{A}$ be some* MR-UMA$_{\mathcal{E},q}$*-adversary. Then we give a* PRF$_{\mathcal{E}}$*-adversary $\mathcal{B}$ such that*

$$\mathbf{Adv}^{\text{mr-uma}}_{\mathcal{E},p_m,q}(\mathcal{A}) \le \mathbf{Adv}^{\text{mr-uma}}_{\mathcal{E},p_m,q}(\mathcal{A}^*) + \mathbf{Adv}^{\text{prf}}_{\mathcal{E}}(\mathcal{B})$$

$\mathcal{B}$ makes q queries and runs in time $\mathcal{T}(\mathcal{A}) + 2q + q \cdot \mathcal{T}(p_m)$.

$$\begin{aligned}
&\arg\max_f \Pr\left[\, C_1, \ldots, C_q \mid f \,\right] \\
&\quad = \arg\max_f \prod_{i=1}^{q} \Pr\left[\, C_i \mid M_1 = f^{-1}(C_1), \ldots, M_q = f^{-1}(C_q) \,\right] \\
&\quad = \arg\max_f \prod_{i=1}^{c} p_m(f^{-1}(\tilde{C}_i))^{N_{\tilde{C}_i}}
\end{aligned}$$

$$\tilde{\mathcal{E}} \colon \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$$

Let Perm$(\mathcal{T}, \mathcal{M}$ be set of all functions $\mathcal{T} \times \mathcal{M} \to \mathcal{M}$ for which

```
$\underline{\mathrm{TPRP1}_{\tilde{\mathcal{E}}}^{\mathcal{A}}}$
$K \leftarrow\!\!\$\ \mathcal{K}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Fn}}$
Return $b'$

$\underline{\mathrm{Fn}(T, M)}$
Return $\tilde{E}_K(T, M)$
```

```
$\underline{\mathrm{TPRP0}_{\tilde{\mathcal{E}}}^{\mathcal{A}}}$
$\tilde{\pi} \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{T}, \mathcal{M})$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Fn}}$
Return $b'$

$\underline{\mathrm{Fn}(T, M)}$
Return $\tilde{\pi}(T, M)$
```

$$\mathbf{Adv}_{\tilde{\mathcal{E}}}^{\mathrm{tprp}}(\mathcal{A}) = \left|\Pr\left[\,\mathrm{TPRP1}^{\mathcal{A}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{TPRP0}^{\mathcal{A}} \Rightarrow 1\,\right]\right|$$

```
$\underline{\mathrm{REAL}_{\mathsf{SE}}^{\mathcal{A}}}$
$K \leftarrow\!\!\$\ \mathsf{kg}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Enc}}$
Ret $b'$

$\underline{\mathrm{Enc}(M)}$
$C \leftarrow\!\!\$\ \mathsf{enc}_K(M)$
Ret $C$
```

```
$\underline{\mathrm{RAND}_{\mathsf{SE}}^{\mathcal{A}}}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Enc}}$
Ret $b'$

$\underline{\mathrm{Enc}(M)}$
$C \leftarrow\!\!\$\ \{0,1\}^{\mathsf{clen}(M)}$
Ret $C$
```

$$\mathbf{Adv}_{\mathsf{SE}}^{\mathrm{ind\$}}(\mathcal{A}) = \left|\Pr\left[\,\mathrm{REAL}_{\mathsf{SE}}^{\mathcal{A}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{REAL}_{\mathsf{SE}}^{\mathcal{A}} \Rightarrow 1\,\right]\right|$$