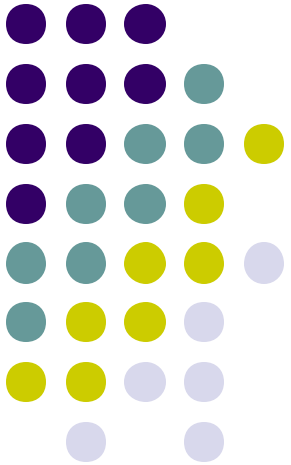
X9.82 Part 1 Overview and Basic Principles

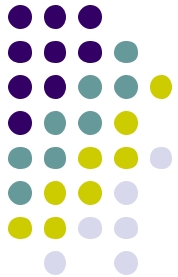
Don B. Johnson

NIST RNG Workshop

July 19, 2004



Random Numbers are Critical



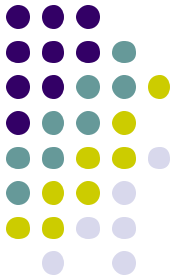
A good RNG: **essential** for secure crypto

- Most papers and algorithms **assume** it!
- It is **far** from trivial to **achieve** it!
- Netscape PRNG seed flaw
- My Grandma's name (used as a PRNG seed) happens to match yours

Quest for a Random Number AKA Hunting the Snark

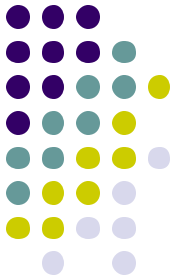
- We want an application to be able to call a random number generator (RNG) and get returned what is called a **random number**, which has certain "magical" properties
- What are those properties, specifically?
- How can we have high assurance that an RNG meeting our specs achieves them?

Learn from Previous Mistakes



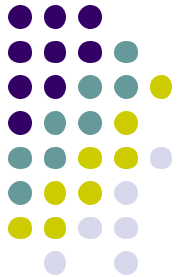
- Review some **“OOPS!”** from the past
- We want to ensure we address these
- Remember that presenting these makes it easy to ‘connect the dots’ and see the mistakes, at the time it was not as obvious (obviously)

Oops From The Past - 1



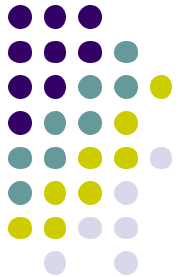
- Previous ANSI X9 PRNG algorithm specifications did **not** specify entropy requirements for the PRNG seed
- Some users thought that **by definition** the output generated by an RNG is a random number, so the seed can be **anything**!
- OOPS!

Oops From the Past - 2



- NIST DSA PRNG outputs a random value **modulo a prime q**
- Bleichenbacher discovers this results in a **skew** in high order bit which leaks info, accumulating such info can break the private key
- OOPS!

Oops From the Past - 3

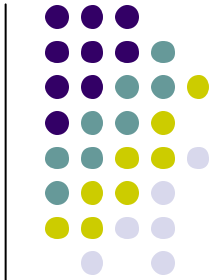


- NIST DSA has an **Approved PRNG** that outputs a random value modulo a prime q
- As it is an **Approved PRNG**, some use it even though they need a random number that is **not** modulo q
- OOPS!

Section 4:
The Importance of Definitions

- Random numbers can be a confusing arena
- People use varying definitions in differing contexts, which if not understood, can lead to further confusion
- After a lot of discussion, X9.82 workgroup composed the following definitions for the purposes of this standard

Def: Random Number



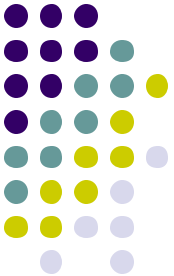
A **value in a set** that has an equal probability of being selected from the total population of possibilities and hence is unpredictable.

A random number is an **instance** of an unbiased random variable, that is, the output produced by a uniformly distributed **random process**.

Def: Random Number Generator (RNG)

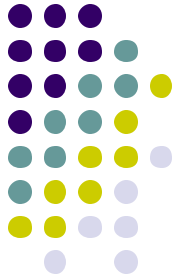
A device or algorithm that can produce a sequence of random numbers that **appears to be** from an ideal random distribution.

Def: Random Bit Generator (RBG)



A device or algorithm that outputs a sequence of bits that **appear to be** statistically independent and unbiased.

14. Converting Random Bits to Numbers and Vice Versa

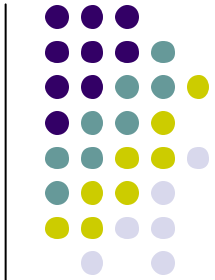


Mistakes were made when one had random bits and wanted a number or had a random number and wanted bits

Specify good deterministic methods to do these conversions

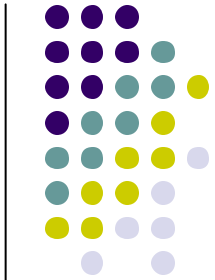
Once specified, can treat all randomness the same, except for the conversion

14.1 Converting Random Bits to Random Numbers



1. Simple Discard Method
   no skew, 1 output
2. Complex Discard Method
   no skew, n outputs
3. Simple Modular Method
   negligible skew, 1 output
4. Complex Modular Method
   negligible skew, n outputs

14.2 Converting Random Numbers to Random Bits



1. No Skew

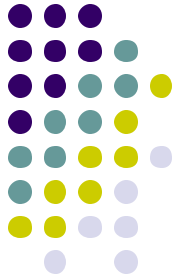
    Variable Length Extraction

2. Negligible Skew

    Fixed Length Extraction

Once these deterministic conversion routines are defined, this aspect of problem is solved

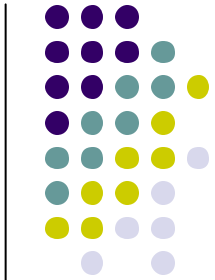
Def: Non-Deterministic Random Bit Generator (NRBG)



An RBG that produces outputs that are **fully dependent** on some unpredictable physical source that produces entropy.

The output of a correctly operating NRBG output has assurance of **full entropy**, contrast with DRBG outputs.

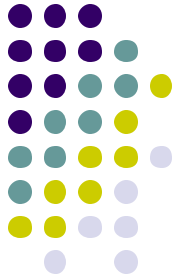
Def: Full Entropy



A bitstring is said to have full entropy when the number of operations that an adversary needs to have at least a 50% chance of guessing that bitstring is determined solely by its length.

An NRBG is said to have information theoretic security, can be considered to have a security level of infinity.

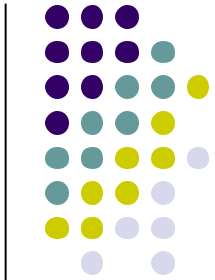
Def: Deterministic Random Bit Generator (DRBG)



An RBG that uses a **deterministic** algorithm to produce a pseudorandom sequence of bits from a secret value called a ***seed*** along with other possible inputs.

DRBG outputs are **not** guaranteed to contain full entropy; contrast this with NRBG outputs.

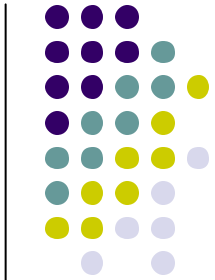
Def: Negligible Probability



A probability of something happening that **can be ignored** as it does not contribute to a realistic attack on the cryptographic mechanism.

The threshold depends on the specific mechanism; it is not < the chance associated with a successful guess & not > 2^{-32}.

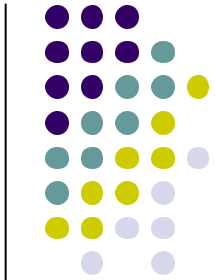
Def: Statistically Unique



A value is said to be statistically unique when it has a **negligible probability** to occur again in a set of such values.

When a random value is required to be statistically unique, it may be selected **either with or without replacement** from the sample space of possibilities.

Section 6: Many Uses for Random Numbers



Symmetric cryptographic keys

Asymmetric cryptographic key generation

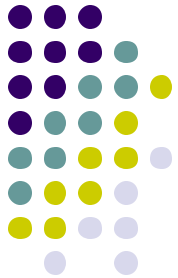
PIN generation

Initialization values/vectors

Challenge nonces in protocols

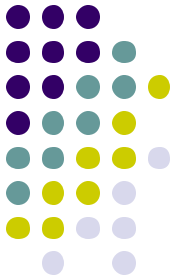
Random seeds to show arbitrary generation of domain parameters

Section 6: Why Random Numbers?



- Game Theory provides the insight that selecting a key at random from the key space means there is no short cut method for an adversary to guess it
- If not random, then an adversary may figure out **how** the key is non-random, effectively reducing the key space
- Q: What would be the **perfect** solution?

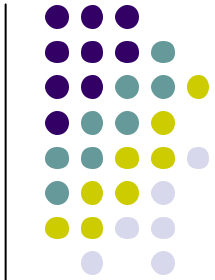
7.1: Idealized Coin Flips Heads or Tails



What are the **Ideal** Properties (ignoring potential for overlap)?

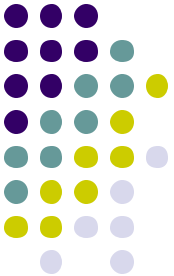
1. Unpredictable
2. Unbiased
3. Independent of previous flips
   A. Backtracking is impossible
   B. Prediction is impossible

What Can Go Wrong?



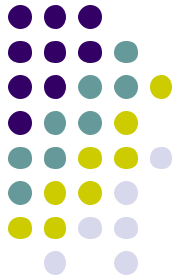
- Predictable – stage magicians can flip a coin in a repeatable way
- Biased – coin skewed to prefer one side over the other
- Dependent – coin can wear down, thumb can get tired
- True story from my life – using a half dollar to determine what to do

Section 7.2
Addressing Bias



- John von Neumann discovered a way to remove bias even when the bias was unknown (but assuming it is stable)
- HT = 1, TH = 0, skip HH & TT
- Yuval Peres made the method multi-level to **extract** the most unbiased bits
- Can also be used to **assess** entropy
- Peres pseudocode in Annex C.1

Section 7.4 & Annex A.4 Entropy – What Kind?



Many “flavors” of entropy:

Shannon entropy: coding for data transmittal

Guessing entropy: guess the whole value

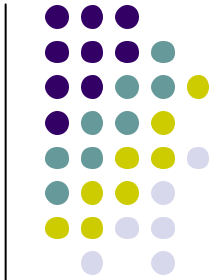
Min-entropy: “worst case” entropy

Each can be added: E(A) + E(B) = E(A||B)

Shannon >= Guessing >= min-entropy

What should X9.82 use as entropy measure?

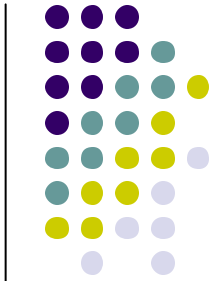
Min-Entropy



- Perhaps surprisingly, the decision was made to use min-entropy
- This gives the highest assurance as it is the worst case entropy as crypto deals with worst cases sometimes
- It is also easy to assess in most cases

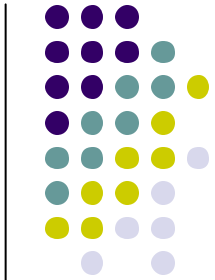
- *** Break ***

X9.82 Ideas



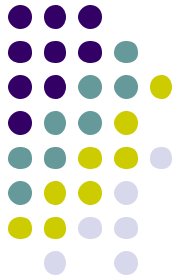
1) Raise the bar
2) Conservative designs
3) Entropy is precious
4) One size fits all?

1) Raise the bar



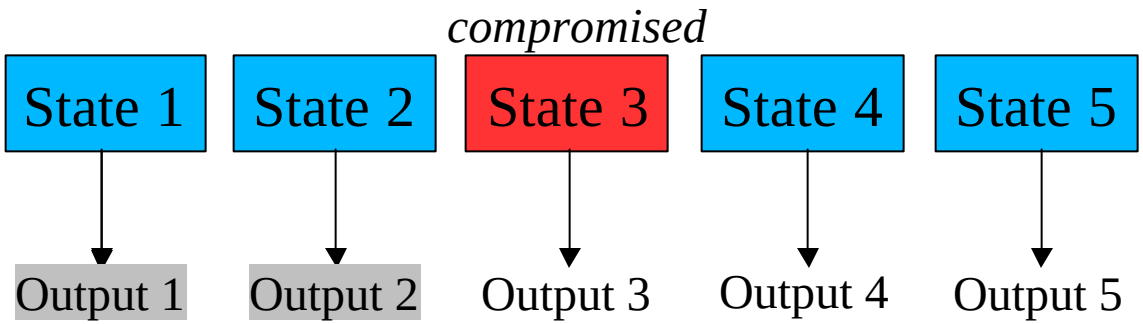
- Take ANSI X9.82 to **the next level**
- Improve existing methods as feasible
- Subjective objective: adversary will **look elsewhere** in cryptosystem for a weakness
- Clearly identify requirement: RNG output is unpredictable and statistically unique
- 2 DRBGs using the same SEED is **not OK**

2) Conservative Design



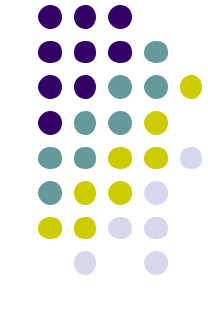
- **Over-engineer** portions as appropriate
- For DRBG x-bit security, use appropriate SEED entropy to avoid internal state collisions
- Mandate DRBG backtracking resistance and allow prediction resistance in case info about a seed is discovered somehow

Backtracking Resistance



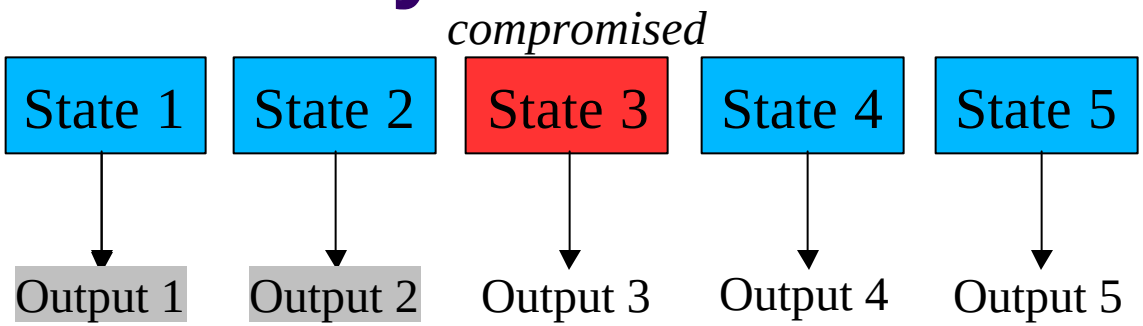
- If someone breaks into your system's internal state today, they cannot go backwards in time to reveal previously generated values.
- Can be achieved by crunching the entropy in the internal state through a one-way function

Def: Backtracking Resistance



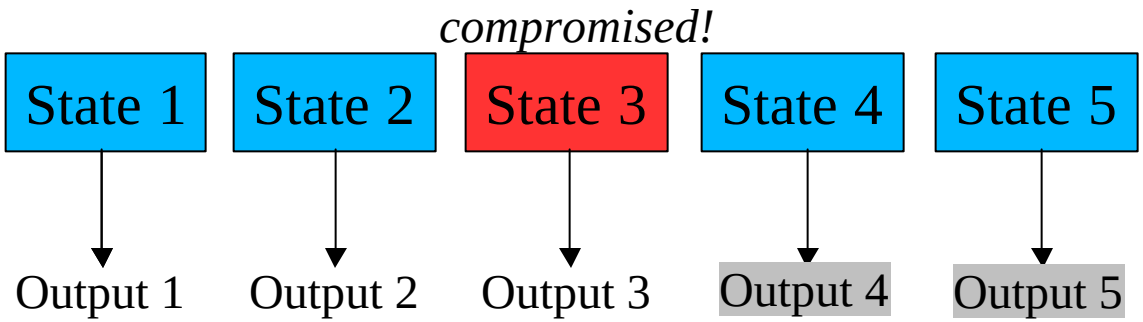
The assurance that an output sequence from an RBG remains indistinguishable from an ideal random sequence even to an adversary who compromises the RBG in the future, up to the claimed security level of the RBG.

Backtracking Resistance & Forward Secrecy



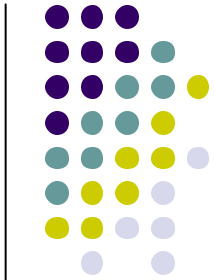
- **Backtracking resistance** is the property needed to achieve **forward secrecy** in a protocol
- Difference in names is due to differing perspectives: in forward secrecy, it is what can I do NOW in case a compromise occurs in future

Prediction Resistance



- If you know internal state now, cannot look forward to predict future output
- Cannot be done without adding new entropy (randomness)
- Note: Achieving full entropy includes achieving prediction resistance and backtracking resistance

Def: Prediction Resistance

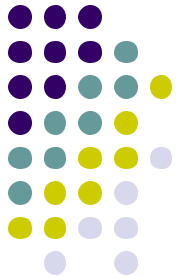


The assurance that the output sequence of an RBG remains indistinguishable (up to the security level of the RBG) from an ideal random sequence to an adversary who has compromised the RBG at some specific time in the past.

3) Entropy is precious

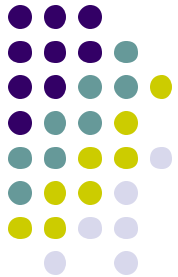
- Obtaining and assessing entropy are challenges to any RNG design
- Once you have entropy, preserve it
- Design the system to avoid dangerous actions that can destroy it

Entropy Assessment



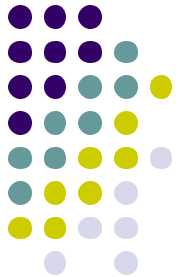
- How unpredictable is the entropy input?
- Ex: Use Peres multi-level strategy to assess entropy via “effective” bit length
- Assumes each bit is independent
- Use raw bits as input as the raw bits may have extra randomness

Statistical Tests



- Statistical tests on a true RNG or on a pseudo RNG should show no difference
- **How many** bits to test?
  - For some 1G is easy, for others 20K is hard
- **What** tests to run? Cost/benefit tradeoff
  - (There is always “yet another possible test”)
- **When** is a deviation OK? A hard problem.
  - On a failure, run again on new “random” bits?
- Testing is necessary but not sufficient!

Properties vs. Requirements

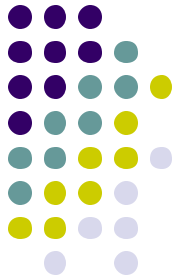


For X9.82, an property is not the same as a requirement:

1) An **Property** describes the ideal to guide RNG product designer

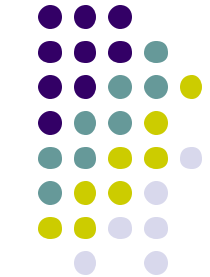
2) A **Requirement** is able to be **validated** through implementation testing

4) Cost/Benefit Tradeoffs



- Any X9.82 RNG **must be good**
- The **assurance of goodness for a product** that is appropriate for a client may **not** be the same as for a CA
- Set minimum tests, allow more as appropriate
- The amount of RNG **output separation** appropriate for a cheap smartcard may not be the same as that for a CA
- Allow output separation, but don't mandate it

Overengineer via Conservative Design

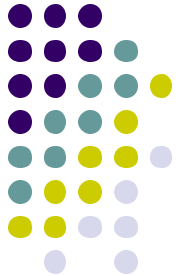


- If at 80 bits of security, losing 1/4 of entropy (60 bit security) means it is **attackable** by a determined adversary; losing 1/2 of entropy (40 bit security) means it is very **weak**
- What are the **minimum requirements**?
- Want **high assurance** that minimum requirements are met, can be exceeded

Attack Models

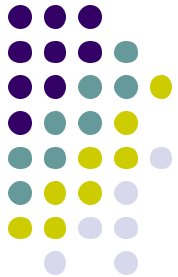
- **Black Box** – adversary cannot peek into insides, the ideal
- **Glass Box** – adversary can see everything, useful in allowing recovery
- **Kerckhoffs' Box** – adversary cannot see the secret inside, real world assumption
- Helps in specification of requirements

Section 8
Properties (Goals) for an RBG



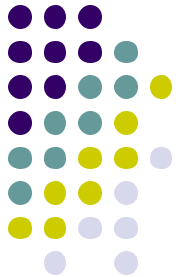
- Cannot distinguish outputs from uniform distribution
- Given only output, infeasible to deduce old bits or predict new bits.
- Outputs are statistically unique
- Collection of outputs pass statistical health tests

Previous Pseudo RNG Properties

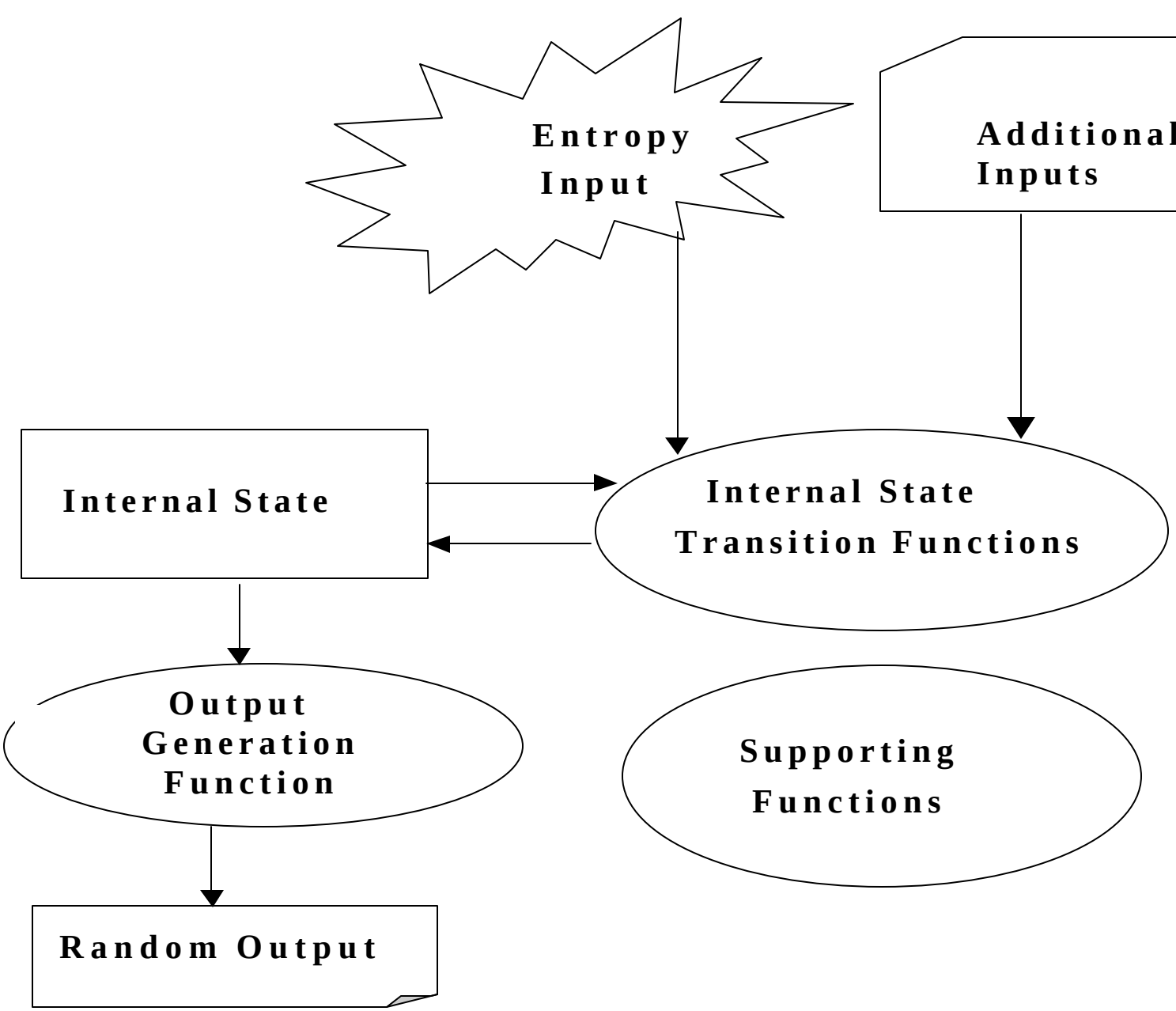


- Previous PRNGs had different properties
- Time variant (ANSI X9.17)
- Allows user input input after setup (DSA)
- Based on hash or block cipher
- NO previous Approved True RNG

Section 9
Unified Functional Model



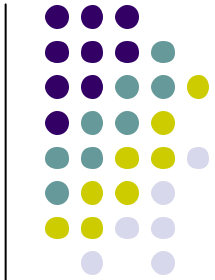
- Unify pseudo and true random number generators into one unified functional model
- See what is common and what differs
- Synergy: if one design meets a requirement one way, how does another?
- Reduce chances of overlooking something


Entropy Input
Additional Inputs
Internal State
Internal State Transition Functions
Output Generation Function
Supporting Functions
Random Output

Entropy Input

- Composed of **varying digitized** bits
- Has been **assessed** to contain at least a certain amount of entropy
- Not guaranteed to have full entropy
- Ex: DRBG seed input, digitized entropy source, output from RBG

Additional Inputs

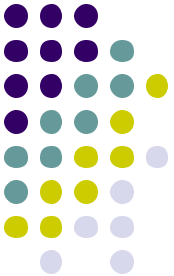


- Secret or non-secret data
- May be used to personalize RBG
- May be used to add variability, but RBG does not depend on it for its security
- Can be null

Internal State

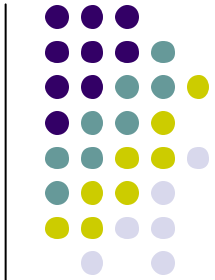
- The memory of the RBG
- Can contain raw input bits
- Can contain processed bits ready for output
- Can keep track of the state

Internal State Transition Functions



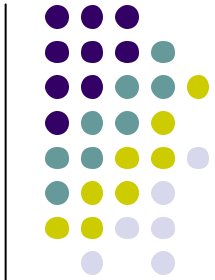
- Changes one internal state into another
- Ex: Peres unbiasing on raw coin flips
- Deterministic, can be tested for correct operation via known answer tests

Output Generation Function



- Sole job is to give random output as a result of a call to the RBG
- Deterministic, can be tested for correct operation with known answer test

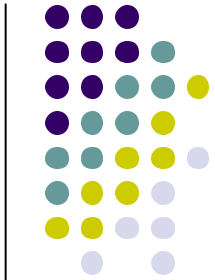
Support Functions



- Any additional function needed that is not an ISTF or OGF
- Ex: Statistical health tests, known answer tests

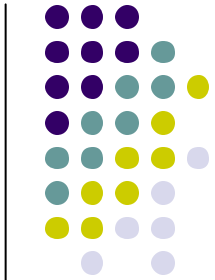
- *** Break ***

Section 11
Conceptual APIs



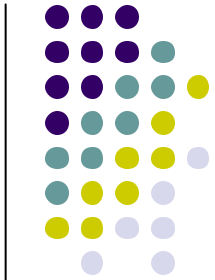
- Specify 2 conceptual APIs for Entropy Input Process call and RBG call
- Helps ensure it all hangs together
- Specifies a clean interface that can be called when needed
- Not required to be implemented in the exact way specified

EIP_Get_Entropy Function



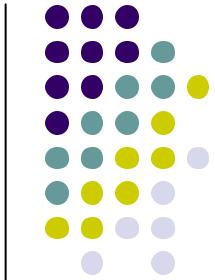
- **Input:** requested bit length of entropy input
- **Output:** entropy input, estimate of min-entropy, status
- **Process:** The entropy source is sampled to obtain the output bits (entropy input) and the assessed estimate of the min-entropy in those bits is also returned, along with an indication of success or failure of the request.

EIP_Selftest Function



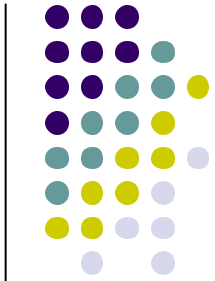
- **Input:** none
- **Output:** status
- **Process:** The self tests that are available to be executed to provide assurance of correct operation are invoked and the status is returned indicating success or failure.

RBG_Instantiate Function



- **Input:** security level, prediction resistance supported flag, full entropy supported flag, personalization string, RBG specific params, mode
- **Output:** handle, status
- **Process:** see following slides

Def: Security Level

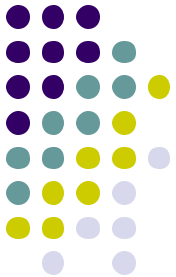


A specific value from the set

(80, 112, 128, 192, 256)

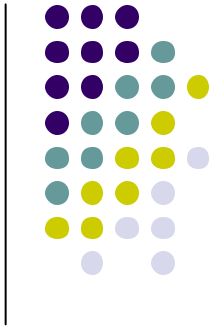
The number of operations needed by an adversary to break a crypto key or algorithm is 2 raised to the security level

NIST/ANSI X9 Security Levels Table



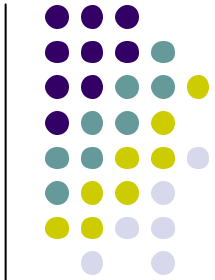
	Exhaustion	Collision
80 bits: to 2010	2-Key TDES	SHA-1
112 bits: to 2030	3-Key TDES	SHA-224
128 bits: 2031+	AES-128	SHA-256
192 bits: 2031+	AES-192	SHA-384
256 bits: 2031+	AES-256	SHA-512

handle



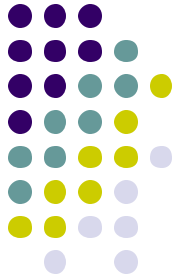
- Acts like an index to an instantiation
- Additional handles may be supported to more finely separate different uses of random numbers.
- Ex: one may wish to separate secret/private output from public output, to separate secret symmetric key use from private asymmetric key use, etc.

RBG_Generate_Random_Bits



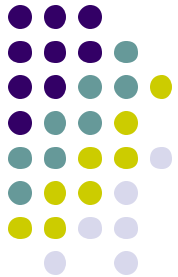
- **Input:** handle, number of output bits, security level, additional input, prediction resistance requested flag, full entropy requested flag, RBG specific parameters, mode
- **Output:** status, random bitstring
- **Process:** See following slides

prediction resistance requested flag



- If the prediction resistance requested flag is set, then output **shall** be inhibited unless and until sufficient entropy has been (possibly already) added, based on the security level specified.
- If prediction resistance is not supported, then an error is returned.

full entropy requested flag

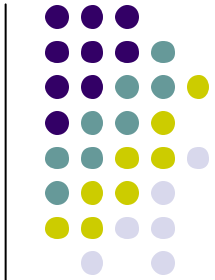


- If the full entropy requested flag is set, then the output **shall** have full entropy. This means that the RBG must be capable of operating as an NRBG; If the RBG cannot provide full entropy, then an indication of failure is returned as the status

mode

- The mode parameter indicates whether normal operational instantiation is being performed, or whether the RBG_Instantiate function is being called as part of a self testing process.

RBG_Selftest

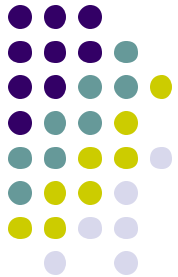


- **Input**: Not applicable
- **Output:** status
- **Process:** This is a Support Function. The deterministic parts of the RBG are tested using known answer tests for correct operation

RBG_Uninstantiate

- **Input:** handle
- **Output:** status
- **Process:** The instantiation associated with the handle parameter is taken down and the resources allocated to it are freed.

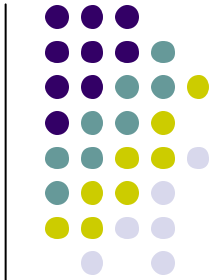
DRBG's: Simple, but not Simplistic



- Specify at least one good DRBG method using:
  1) hash function
  2) block cipher
  3) ECC
  4) RSA

More details later in Part 3 presentations

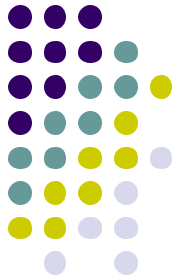
Types of NRBGs



- Basic – does **not** have an Approved DRBG as a backup
- Enhanced NRBG – has an Approved DRBG as a backup

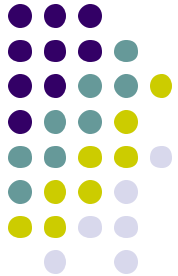
- More details later in Part 2 discussion, but some introduction here

Basic NRBG Risks



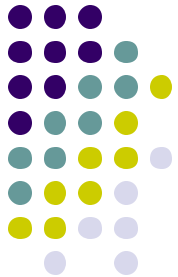
- An NRBG can degrade over time
- Parts can wear out, can operate outside its environmental range, can be attacked physically
- Use of a raw NRBG by itself may be **risky!**
- Need to detect quickly when outside of normal operation

Enhanced NRBG Mitigates Risks



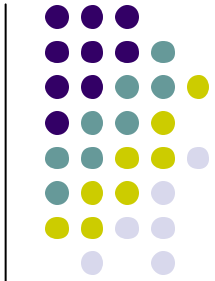
- Combine NRBG with validly initialized DRBG. This is a **safe default** solution
- Degradation of true random bits is masked by a fall back method
- If NRBG degrades even to **zero entropy**, may still be secure up to backup security level of DRBG
- **Conservative design**! High assurance

Section 12.4
RBG Cost/Benefit Spectrum



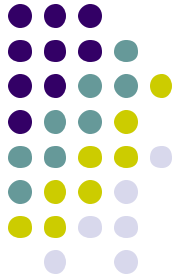
- DRBG with a fixed seed
- DRBG with manual reseed capability
- DRBG with automatic reseed capability
- Enhanced NRBG designed as a DRBG with continuous reseed capability
- Enhanced NRBG with full algorithm independence

Section 12.5
RBG Cost/Benefit Choices



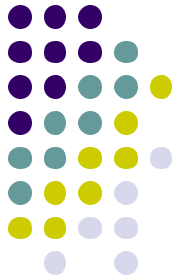
- Symmetric Based DRBG
- Asymmetric Based DRBG
- Basic NRBG
- Enhanced NRBG

Section 13.1
Merging an NRBG with DRBG



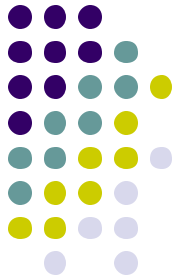
- A merged design is one which includes both an NRBG and a DRBG
- Each meets its own requirements
- DRBG: very fast
- NRBG: high assurance
- Merger can be best of both worlds

Section 13.2
Combining RBGs



- Allow **combining** 2 or more RBGs via **XOR** (Boolean bitwise Exclusive OR) for increased assurance
- Sanctions combining a **non-approved** method with an Approved one
- Assumption is the non-approved RBG is better in some way, yet not Approved

Summary of X9.82 Part 1



1. Standardized Definitions
2. Properties/Requirements Distinction
3. Unified Functional Model w/ Requirements for Components
4. Specified Conceptual APIs
5. Cost/Benefit Choices
6. Approved Combining/Merging RBGs
7. Approved Conversion Routines