SECWORKS

CRYPTECH

Cryptech
TRNG Ideas 2014-03-17

Joachim Strömbersgon – Secworks AB

SECWORKS

CRYPTECH

The Cryptech TRNG

A state of the art, modern
True Random Number Generator
that can be verified by the user

TRNG – High Level Requirements

- Development drivers
  - Good performance – in embedded, low cost systems
  - Conservative design - No wild, untested ideas and algorithms
- Support transparency and testability
  - Debug access to raw entropy. Debug access to CSPRNG with injection
  - All normal access blocked when operating in debug mode
  - Full restart including warm-up when leaving debug mode
- Configurable and scaleable
  - Security (number of rounds),
  - Number of entropy sources
  - Amount of entropy/seed value
  - Amount of Random values/seed

With sane/conservative defaults

TRNG – Threats and Mitigation (1)

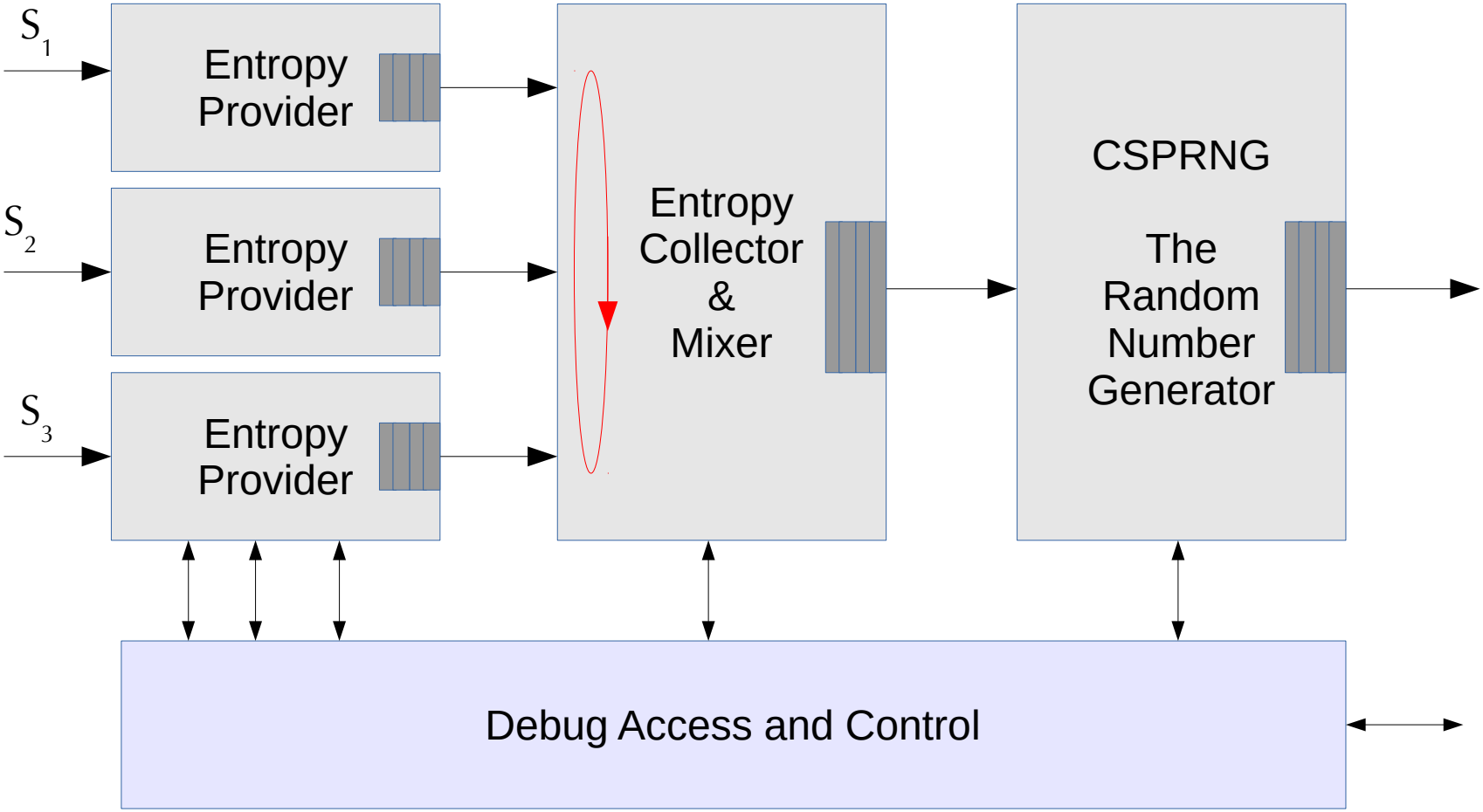
- Manipulation of entropy source
  - Support multiple and different entropy sources to force an attacker to manipulate more than one physical process simultaneously
  - Continuously observe the entropy sources to determine their health
  - Support access to raw entropy for off-line testing
  - Use cryptographically good mixing of entropy source to make it hard for an attacker to predict the effect o manipulating an entropy source.

TRNG – Threats and Mitigation (2)

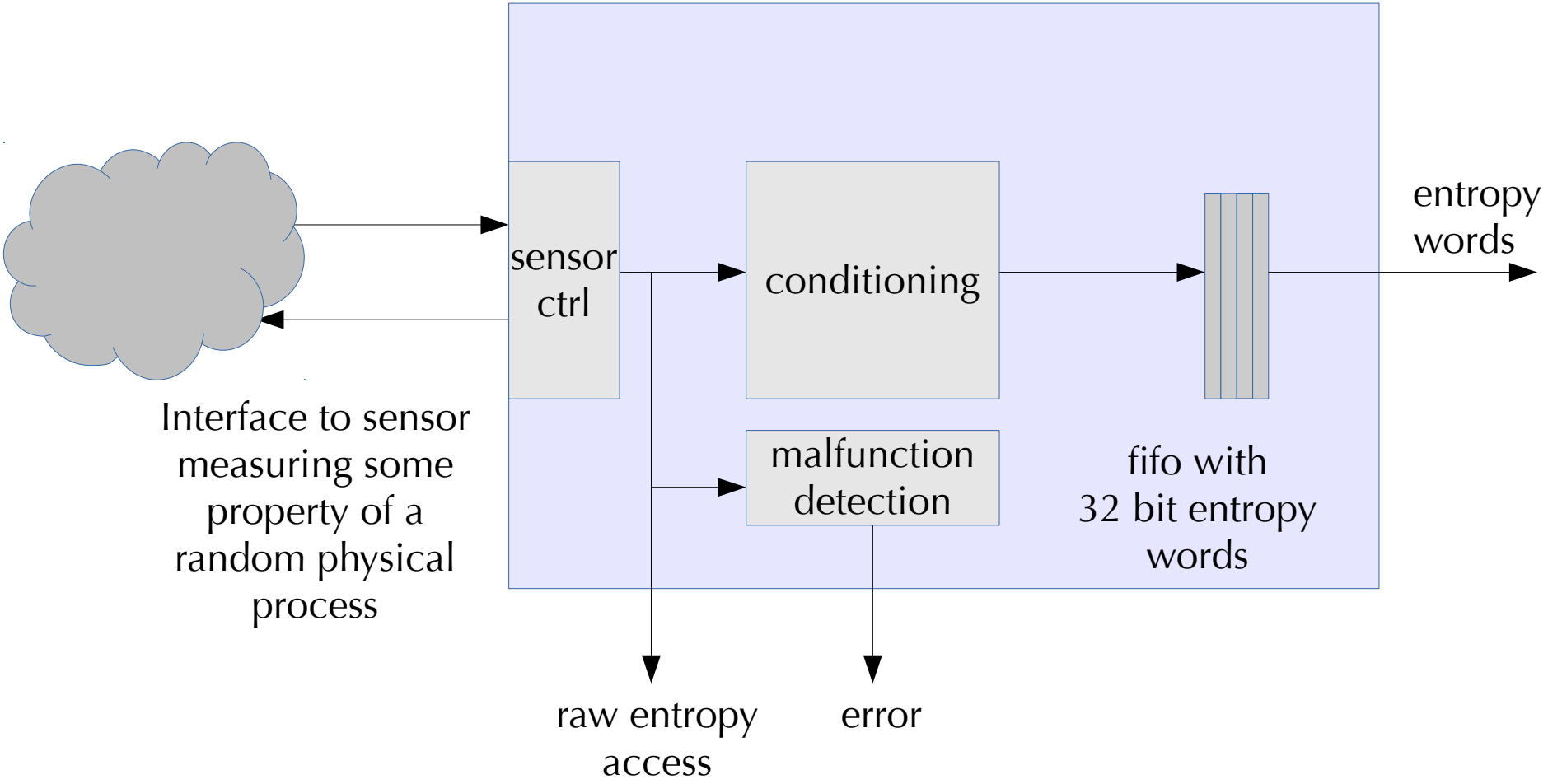
- Loss of seed driving the CSPRNG
  - Periodically reseed to minimize impact
  - Increase cost of brute forcing the seed
  - Use a mixer that makes it infeasible to determine entropy values that generated the seed

- Denial of Service - Random Number Starvation
  - Use high performance CSPRNG to generate random numbers.
  - Support scaleability of using parallel CSPRNGs
  - Fast warm up using stored random numbers as entropy

TRNG - Architecture



Entropy Providers (1)

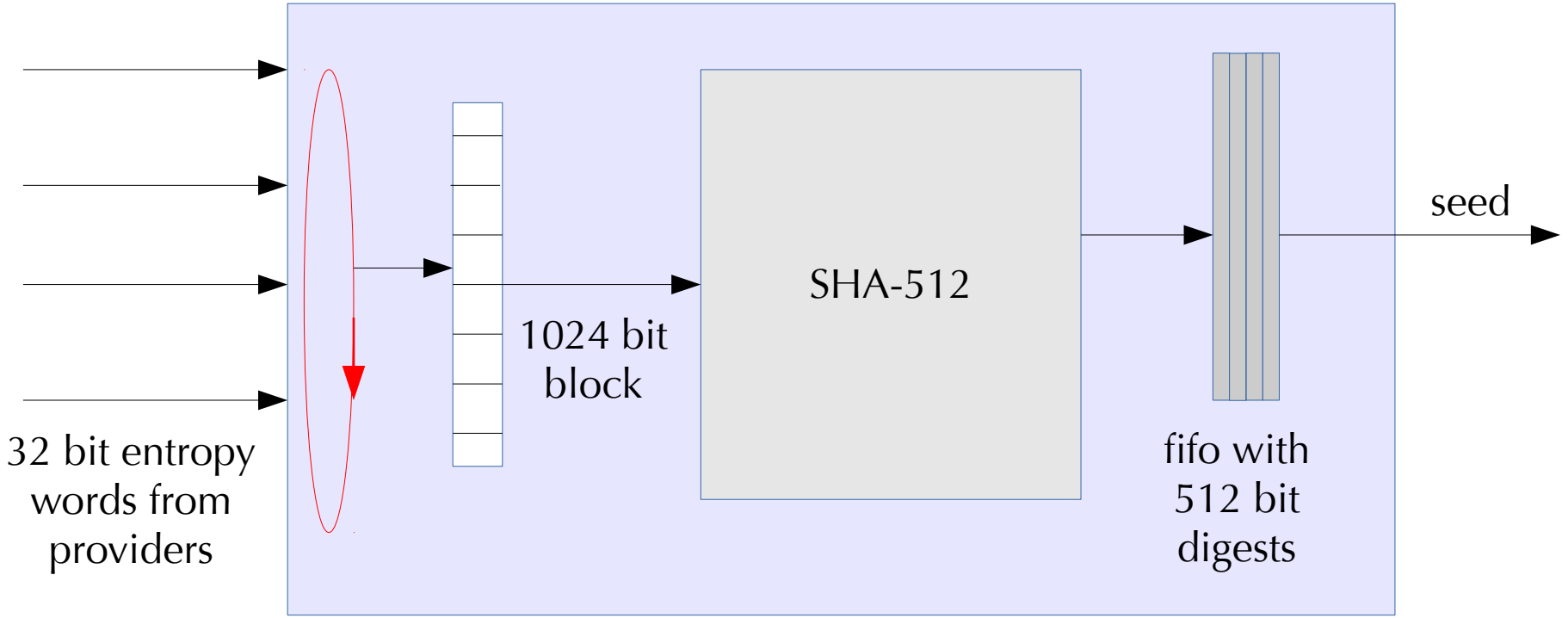


Entropy Providers (1)

- Access to entropy sources
  - Acts as interface for a given source. Hide specifics from rest of TRNG
    - P/N reverse bias avalanche noise
    - CCD black out noise
    - RSSI LSB
    - Oscillator jitter
    - etc
- Detect malfunctioning entropy sources
  - on-line testing to detect a broken source
- Condition, whitening of entropy
  - Remove bias

Stored random values for fast warm up are treated as just another entropy source with a provider

Entropy Collection and Mixing (1)



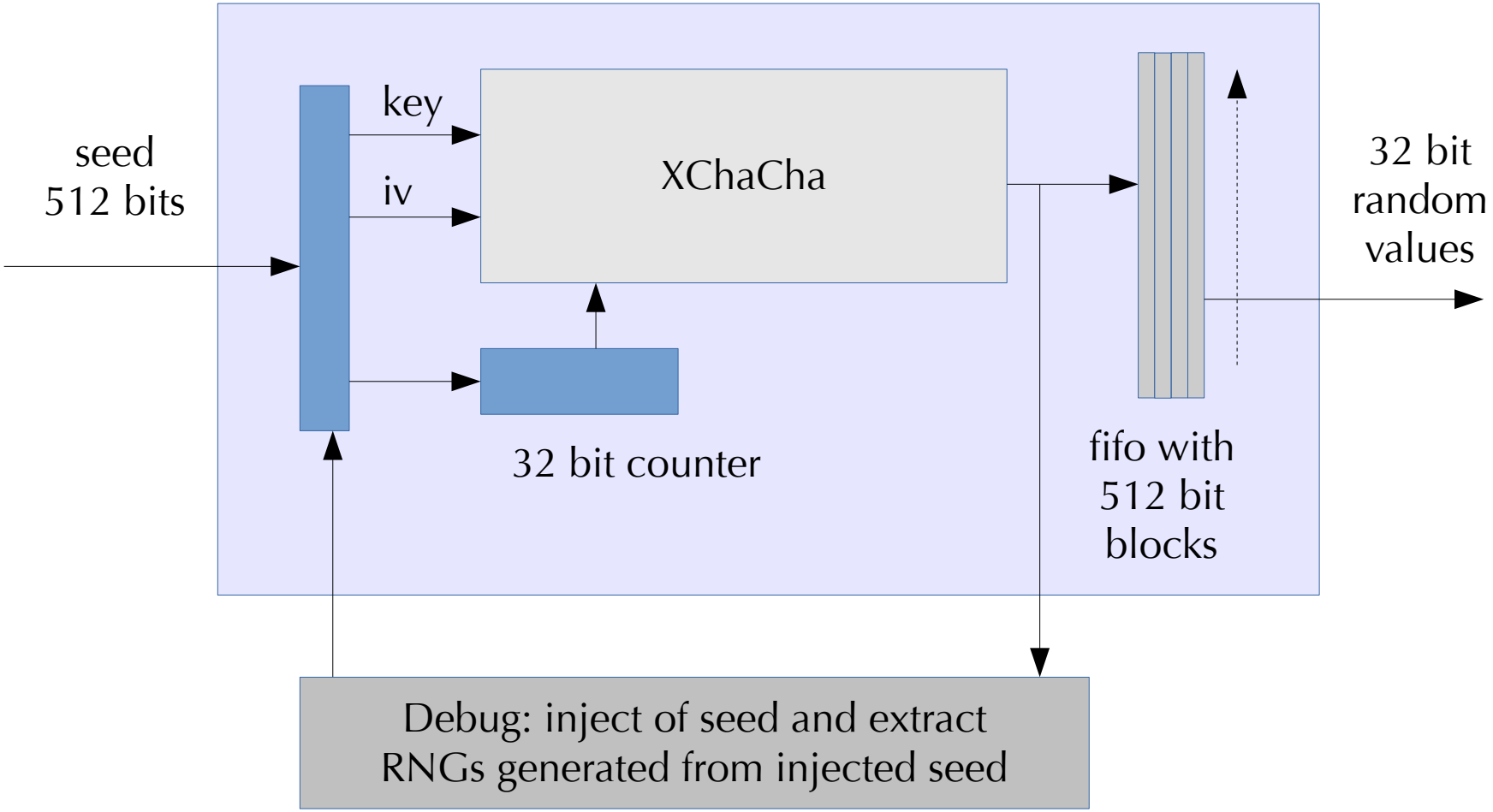
Entropy Collection and Mixing (2)

- Combine entropy from different providers
  - Round robin retrieval to avoid starvation of providers with lower rate
- Generate high quality seed values
  - High grade mixing - bias free seeds
- Robust against attacker induces bias
  - Infeasable to predict effect of induced bias on seed by attacker

Entropy Collection and Mixing (3)

- 32 entropy words form a *message block*
- Message block fed to SHA-512 (FIPS 180)
- n message blocks processed in sequence
  - n can be configured
- 512 bit digest from SHA-512 delivered to RNG as seed

Random number generation (1)



Random number generation (2)

- Generate high rate of random numbers
  - In embedded systems with low clock frequency
- Generate random numbers with good quality
  - Cryptographically strong CSPRNG
- Low impact of reseeding
  - Short latency for initialization

And support testing too!

Random number generation (3)

- Based on XChaCha with 256 bit key
  - Scalable security (number of rounds)
  - Scalable performance
  - Not AES, but fairly well proven. Good traction

AES-CTR would be the alternative

- 96 bit IV and 32 bit counter
  - (2**32 – 1) keystream blocks between reeseding
  - 255 Gbyte max from a single seed

- At least 24 rounds. 32 possible with good performance

Seed used for key, IV, counter start

Random number generation (4)

- Debug support for testing
  - Inject user defined seed
  - Generate at most n blocks of random values then stops
  - Access to generated values via debug port only
  - Reading from normal port during debug yields is a read error.
  - Full restart including warm up when leaving debug mode
- Generates random values continuously
  - Overwrites blocks in output buffer unless being used
  - Some random values are sent to secure storage as entropy for fast warm up.

References and inspiration (1)

- /dev/random in FreeBSD and OpenBSD
  - Fortuna, Yarrow
  - ChaCha as CSPRNG
- /dev/urandom in Linux
  - Mixer
- The Fortuna RNG/collector
  - Handling of multiple entropy sources and mixing
- IanG Hard Truths about the Hard Business of finding Hard Random Numbers (http://iang.org/ssl/hard_truths_hard_random_numbers.html)
  - Architecture, multiple sources etc

These are just a few references.