Limitation:

RSA Encryption Attack requires attacker to have some knowledge over the plaintext for it be effective. Taking the lab example, if the plaintext is not only a single number (e.g. 100) but is something like a HTTP request with lots of content. The attacker might have a hard time figuring out the exact section where he would insert his own message to forge the entire request.

RSA Digital Signature Attack is not useful because it is difficult for attacker to find a meaningful preimage of a hash. While attacker can generate as much random (hash,signature) pairs as he wants, the receiver will only proceed to verify the signature only if the received message actually make senses to him.
For example, suppose the attacker computed a hash from a randomly generated signature s and he wants to trick Alice into sending him $1,000,000.
He will have keep generating different versions of message that in one way or another asks Alice to transfer $1,000,000 to him.
For the sake of demonstration,
message version 1 can be "Send Attacker $1,000,000"
message version 2 can be "Transfer Attacker $1,000,000"
message version 3 can be "Transfer Attacker $1,000,000.00"
…

and the list goes on.
Also, there is no guarantee that attacker will be able to a version that Alice will accept as valid because maybe Alice is looking for a certain kind of message semantic such as format.

----------- Optimal Asymmetric Encryption Padding (OAEP) -----------

The purpose of OAEP is to break the deterministic property of Textbook RSA by introducing randomness. With the usage of a nonce, the same message will be encoded into different string every time and therefore RSA will encrypt into different ciphertext. The security lies with the hash functions used.

===Algorithm===

Parameters:

- n is the number of bits in RSA modulus
- G and H are cryptographic hash functions,
   G expands k0 bits to (n-k0) bits
   H reduces (n-k0) bits to k0
- k0 and k1 are two predefined integers based on protocol
- r is nonce
- m is message

Encoding process:

1. m is padded with k1 zeros such that (m || k1*0) is (n-k0) bits in length

2. r is a randomly generated k0-bit string by sender

3. X = (m || k1*0) $\oplus$ G(r)

4. Y = r $\oplus$ H(X)

5. encoded_m = X || Y

Decoding process:

1. Receiver retrieve r by

   r = Y $\oplus$ H(X)

2. Then retrieve m by

   X $\oplus$ G(r) = (m || k1*0)

   Then remove the k1*0 to get back m

----------- Probabilistic Signature Scheme (PSS) -----------

Reference: https://www.cryptrec.go.jp/exreport/cryptrec-ex-1011-2001.pdf

The purpose of PSS is to introduce randomness directly in the signature generating process with salt. PSS also specify protocol to remove randomness from signature.

===Algorithm===

Parameters:

- MGF is Mask Generation Function, SHA-1 is normally used in real life
- s is randomly generated salt (with variable length)
- pddg1, pddg2, bc are fixed padding specified in the protocol
- m is message
- H is a cryptographic hash function

Encoding process:

1. s is generated

2. Form a string M' by concatenating pddg1, hash of message, and salt

   M' = (pddg1 || H(m) || s)

3. Compute hash of M' using H

   hashM' = H(M')

4. Concatenate pddg2 and s to form a data block DB

   DB = (pddg2 || s)

5. Apply MGF on M'

   dbMask = MGF(M')

6. XOR dbMask with DB

   maskedDB = dbMask $\oplus$ DB

7. encoded_m = (maskedDB || hashM' || bc)

Decoding process:

(Reverse of the algorithm)

1. Retrieve DB by

   DB = hashM' $\oplus$ maskedDB

2. Retrieve s from DB, removing pddg2

3. Form back M'

   M' = (pddg1 || H(m) || s)

4. Check if H(M') matches hashM'