[0030] Consistent with the enclave threat model, it can be assumed that the attacker cannot observe the contents or the access patterns to the trusted memory, and cannot tamper with the trusted code implementing the filtering algorithm. While this assumption may hold for certain platforms such as the Sanctum enclave, oblivious memory techniques can be used to hide access patterns to trusted memory in the SGX-based library implementation. On the other hand, the attacker may be capable of full observation of access patterns and contents of unprotected memory; in other words, the attacker may observe the timing of each input consumption and output production at a fine-grained clock granularity.

[0031] The input and output streams can be encrypted under a symmetric key known to the client, and provisioned by the client to the trusted module (e.g., sent over an authenticated channel established using remote attestation). The goal of oblivious filtering is to hide which records from the input stream matched the filter, which would leak some information about the plaintext input. In a vulnerable implementation, the trusted module iterates over the entire input stream, fetching one record at a time, and produces each matched record in the output stream. To prevent ~~obvious~~ oblivious attacks, the trusted module may re-encrypt the matched record before writing to unprotected memory. However, this alone is insufficient, as the attacker simply needs to check whether an output was produced during the time duration between consecutive inputs.

[0032] Oblivious filtering can be used to prevent a (polynomial time) adversary from creating a correlation between the inputs and outputs, in that the attacker may only be able to learn the size of the output stream and nothing more. The filter can temporarily store matched records in the trusted memory buffer, and evict items based on a technique which minimizes leakage while respecting constraints such as buffer size and order preservation.

Data Streaming and Information Leakage

[0033] The input to the system can be a data stream, where each element of which have include some (encrypted) content and a timestamp associated with it. In some embodiments, the timestamp of the data input element need not be considered as part of the content. Rather, it can be assumed that timestamps are publicly readable attributes that are immutable (or cannot be modified). The encrypted

appropriately so that the buffer does not overflow (with high probability), at time t, the expected buffer size is $\mathbb{E}(B_t) = \sum_{i=0}^{t-1} \left(1 - \frac{p_e}{2}\right)^i \left(p - \frac{p_e}{2}\right)$. Since the buffer contains only matched elements, the delay introduced by the element that arrives at time t is the same as the time it takes for at least $\mathbb{E}(B_t)$ evictions to take place. The expected number of outputs produced at time t is at most p. Hence, it takes at least $\mathbb{E}(B_t)/p$ time steps of delay. Moreover, since every eviction causes at least 1 element to evict, the maximum delay for the element at time t is $\mathbb{E}(B_t)/p_e < 3p/p_e^2$. Setting $p_e = \frac{10}{b+2}$, the maximum expected delay is $3p(b+2)^2/100$. Thus, to tolerate Δ, we need $b = \Omega(\sqrt{\Delta})$. From the lower bound, we know that the leakage in this case is $\Omega\left(\frac{1}{\sqrt{\Delta}}\log\Delta\right)$ bits of information per element.

ORAM-Assisted use of Unprotected Memory

[0116] The lower bound discussed above provides that the Collect-and-Evict scheme may have the optimal leakage in the threat model, given a buffer size. However, from Corollary 2, there is a tradeoff between the buffer size and the length of the input stream processed to the extent that a low leakage comes at the cost of potentially very large buffer sizes. The algorithms discussed so far have assumed that the buffer is a trusted memory module, which makes it impractical in reality to offer low leakage since side-channel free trusted memories can be expensive and an overkill if used just for filtering purposes.

[0117] In this section, an optimization of the Collect-and-Evict scheme from above will be discussed. An oblivious-RAM (ORAM) is used to substitute large trusted memory with unprotected memory. The construction, according to some embodiments, makes the size of the trusted memory only a function of the size of elements in the input stream and independent of how many input elements need to be processed. Since unprotected memory is easily available and lower cost, the leakage can be made arbitrarily close to the lower bound, depending mainly on the constraints on the latency of the system. The details of the construction are as follows.

[0118] Let τ be the size of an encrypted element from the input stream and B, M be the size (in bits) of the trusted and unprotected memory (ORAM),