Azure SQL Database Always Encrypted

Always encrypted(AE) is a column based encryption feature by Microsoft that was introduced in Microsoft SQL server. The customer can either install it on their own infrastructure or use it as a PaaS. It had two iterations, first in 2016 which supported equality operations such as filtering and searching over the encrypted columns and used a deterministic encryption. The second iteration introduced the concept of enclave in the TEE. Infrastructure as a service(IAAS) is a popular option but can have certain concerns especially when it comes to security especially if it is health related information, defense or social security number. It is not designed for encrypting the entire database. The goal of AE was to protect the cloud database from rogue employees owning the actual infrastructure hence eliminating this attack vector. Current system supports data-at-rest but are vulnerable to attacks where the data is in-memory. The AE assures that the columns marked as sensitive are always in the encrypted form except when it is inside the TEE. The clients follow the “bring your own key” model and protects from memory scraping attacks. The AE uses the TEE to temporarily store the encrypted key. For the encryption, it uses a two level: A CEK and CMK. The CMK is stored in a separate key provider such as Azure key vault, Java key store and key stores rooted in Hardware Security modules. It currently uses RSA(RSA_OAEP) but the DDL syntax is extensible for CEK encryption and data encryption. Section 3 and 4 focuses on the architecture and its implementation. My key take away would be that AE focus on operational data confidentiality rather than semantic security, uses the enclave only for expression evaluation small tcb are beneficial, a property to connection string helps in avoiding two round trips to SQL, the authors wrote a small SQL OS layer for running expression service(ES) inside enclave, leverages the CTR in 2019 for recovery purposes but if the key is not supplied by the client, certain corner cases such as log truncation and restoring a backup can be an issue. There’s a enclave worker thread in the core, if its busy it avoids context switch, if its doing nothing, it sleeps. Hence, against a strong adversary, the AE system provides a strong protection.

Comments:
In the related work, oblivious ram was mentioned. I don’t think it is again mentioned in paper but will that not provide more resilient against side channel attacks? I agree that hardware based ORAM can be costly but I guess there are software based ORAM which are opensource/less costly in the grand scheme? I am guessing adding one more layer of ORAM can have some hit in performance but for highly sensitive data, that makes more sense and it can be added as an optional feature?
I also wished they talked about the side channel attacks in more detail in this paper and not just briefly in the threat model.
In section 2.1, it is mentioned that the host uses dll to invoke enclave. Since dll is

windows specific, is it just restricted to the windows ecosystem or are there any supports available for the linux(.so file) since lot of infrastructure are on linux based kernels.
Also what happens if the client with the key only gets compromised? Does that Make byok model compromised?
Section 2.4 mentions the functionality that are possible on encrypted data. But lets say I wanted to perform more operations on the encrypted columns, will I not be able to encrypt the columns? Isn't that kind of a major drawback.
Section 2.5 mentions that it only supports parameterized queries due to security reasons. I guess that's a reasonable maintenance overhead but I guess acceptable if security is high priority.